Open Arcade Architecture Device (OAAD) API Specification

Rev. 1.100
October 30, 1998
By: Burt Bicksler and Christopher D. Watkins
Industrial Mindworks, Inc.
http://www.industrialmindworks.com/

Please send comments via electronic mail to Mark_Atkins@ccm.intel.com

Revision History:

0.001	03/15/98	Initial Version
0.002	03/16/98	First Revision
0.003	03/17/98	Updated Version number description
0.004	03/19/98	Updated Q/A issues
0.005	03/21/98	Integrated changes from Steve McGowan(Intel) and some additional formatting clean ups
0.006	03/22/98	Some additional minor cleanups.
0.007	03/23/98	Added copyright block
0.008	03/23/98	Removed Joystick references from GetProperty, added Functional Block diagram
0.009	03/30/98	Revised block diagram, changed DWORD return on GetVersion to HRESULT added method for performing hardware test.
0.010	04/06/98	Added additional implementation description and additional API definitions, as well as changes related to making OAAD able to be used without the presence of DirectInput
0.011	04/08/98	Added appendix for GUIDs and data structures. Cleaned up some of the newly added method descriptions.
0.012	04/10/98	Removed reference the Hardware Compliance testing. Cleaned up some error code names.
0.013	04/12/98	Moved some methods to OAADDevice from OAAD, since that was where they really belonged.

0.014	04/17/98	Cleaned up some grammar, and revised some structure names to be consistent with the implementation.
0.015	04/25/98	Added methods for Watchdog timer support. Added description of physical and logical devices. Added section discussing the concept of physical to logical mapping of devices, and initial data definitions for some of the key devices used in games. Added definitions of key data structures, defines and typedefs.
0.016	04/26/98	Added diagram for physical and logical devices, cleaned up the error code section. Added description of polled and event notification.
0.017	04/29/98	Added Table of contents (sbm).
0.018	04/29/98	Merged in sample application document, cleaned up some formatting, changed bAuto parameter on SetWatchdog to a DWORD specifying the action to be taken when the timer fires(bbb).
0.019	05/03/98	Some general cleanups for format.
0.020	05/04/98	Removed Assumptions section, some other cleanups.
0.021	05/14/98	Some general cleanups.
0.022	05/15/98	Some more cleanups and clarifications.
0.023	05/27/98	Added hyperlinks and additional cleanups.
0.024	06/02/98	Added more descriptions for event notifications, and how to use them in arcade applications.
0.025	06/08/98	Added another level to TOC for individual APIs. Also a number of grammatical corrections.(bbb)
0.026	06/09/98	Some additional minor grammatical corrections. (bbb)
0.027	07/12/98	Added expanded description of the purpose of the OAAD Library. (bbb)
0.028	07/29/98	Cleaned up data structure definitions and more detail to the EnumObjects description. Revised the introduction section and added an implementation sub-section. Added sections on enumerating objects and data formats in the example sections.(bbb)
0.029	09/12/98	Release candidate (bbb)
0.030	09/21/98	Release candidate. revised OADEVICEOBJECTINSTANCE data structure to add a ReportID member (bbb)
1.000	10/05/98	Release 1.0. Revised GetDeviceData descriptions. (bbb)
1.100	10/30/98	Release 1.1. Added new methods for extended events. (bbb)

Table of Contents

1

Introduction

This document describes the Open Arcade Architecture Device Library. The OAAD Library is a collection of functions and data formats that simplify the task of developing arcade applications that interface with a wide range of hardware devices.

The OAAD Library is comprised of two basic components: the OAAD Object, and the OAAD Device Object.

The OAAD Device Object that is implemented in the OAAD.DLL allows the OAAD Library to be extended to support new hardware. An OAAD Device Object is created for each OAAD Device that is supported.

The OAAD Object provides a Microsoft COM compliant implementation mechanism to enumerate the OAAD Devices that are installed on the target system. In addition the OAAD Object is used to create an instance of an OAAD Device Object which the arcade application interfaces with.

The OAAD Device Object shields the arcade application developer from the differences between various I/O devices. The developer needs only to write the application to access the OAAD Library, enumerate OAAD Devices, and create an instance of the desired device. Data access is provided in a generalized manner, which allows for simplified support for new devices.

An application that is developed to use the OAAD Library does not need to be concerned with what specific device it is connected to, only that the device supports the minimum level of functionality that is required by the application.

OAAD will function within the Windows 95*, Windows 98* and Windows NT* 5.x platforms running on Intel architecture PCs. OAAD is designed to work alongside Microsoft's DirectX* (especially DirectInput* and DirectDraw*) version 5.0 and later. OAAD provides functionality and support for devices that are not currently provided by DirectX. This includes extending the limited output functionality of DirectInput and adding support for generalized input/output, coin boxes, debit cards, and similar devices. OAAD abstracts the hardware from the application by providing query and configuration methods to provide as much hardware-independence as possible.

1.1 Conventions

The OAAD Library is implemented in a user-mode Win32 DLL and conforms to the Microsoft Component Object Model (COM) interface specification, as implemented in Microsoft's DirectInput. COM provides a standardized way to create modular software that is easy to extend.

More information on COM is available at http://www.microsoft.com/msdn/library/ under Specifications/Technologies and Languages/Component Object Model.

OAAD is designed to work with Microsoft DirectX. OAAD references the Windows Registry for configuration parameters.

Any software development tool that can generate code to call COM interfaces can be used to create applications that use the OAAD Library. This includes the 'C' and 'C++' languages. Sample code is provided showing access from simple C++ and 'C' applications.

The general input/output data formats used for OAAD devices are based on the Human Interface Device (HID) specifications associated with Universal Serial Bus (USB) devices more information on HID and USB is available at http://www.usb.org/developers.

HID devices support three access methods, called *pipes* in HID nomenclature: Input, Output and Feature. Reports are blocks of data that are passed over pipes between the host and the device. HID also allows a device to generate multiple formats for the reports on an individual pipe. Various Reports in a pipe are distinguished using Report ID's. A device uses an Input pipe to send reports, asynchronously to the host. A device uses an Output pipe to send reports to a device. A Feature pipe is used by a Host to send and receive reports to a device.

DirectInput assumes that a device only supports a single report format on an Input pipe. OAAD addresses this with the OADATAINFO structure, which defines the wReportID member to identify the pipe that a report is associated with and the wReportType member to distinguish a report format on that pipe.

1.2 Implementation

The OAAD Library objects abstract the hardware from the application by providing query and configuration methods to give as much hardware-independence as possible. Methods to obtain information about devices, objects on the devices, data formats and the ranges for various inputs and outputs are provided.

The OAAD Object and OAAD Device Objects are implemented as Microsoft COM compliant objects. Macros are defined that allow the C++ COM interfaces to be called from a program written in the 'C' language, as well as from any other language that supports Microsoft COM interfaces.

The OAAD Device Object that is contained within the OAAD Object provides support for Human Interface Devices on those operating systems that provide full access to HID devices. This is done using the HID Class support present in Windows 98, and Windows NT 5.0, when released.

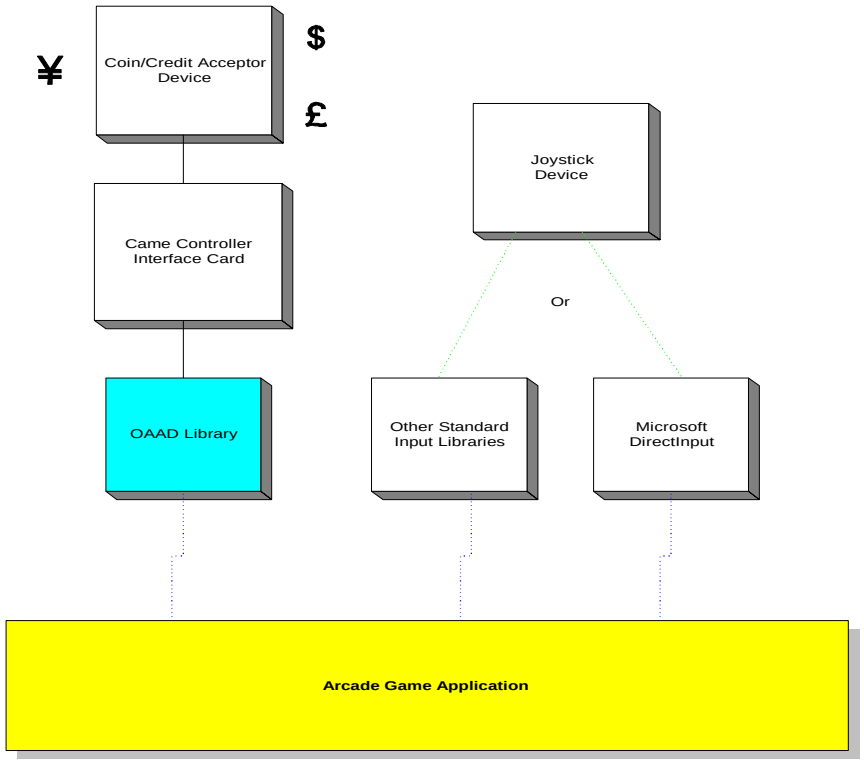
For non-HID devices under Windows 98 and Windows NT 5.0, and for all devices under Windows 95, support is provided by custom OAAD Device objects that encapsulates the physical device that provides the emulated HID functionality and data reports. The manufacturer of the device will typically develop the non-Human Interface Device OAAD Device Object.

All OAAD Device Objects data access is provided in a generalized manner, which allows for simplified support for new devices.

This architecture allows the OAAD Library to be extended to support different, non-HID, I/O devices by developing OAAD Device Objects for them.

For example, the initial OAAD Library includes Device Objects for the Quantum 3D GCI2 and Happ CIB-1000 I/O cards. But it is possible to develop an OAAD Device Object that wraps a different I/O card, such as a generic data acquisition board. Another example would be an application where the coin or bill acceptor interfaces to the application using the serial communications port on the PC. An OAAD Device Object could be developed that encapsulates the serial communications connection to the device. In all of these cases the arcade application would not need to be changed to use the new device.

1.2.1 Functional Blocks



2

Physical and logical devices

The devices an arcade application must deal with fall into two general categories:

- Physical – E.g. Joystick, Game Controller Interface (GCI) card, coin door, etc.
- Logical – E.g. Right Kick Button, Left Punch Button, etc.

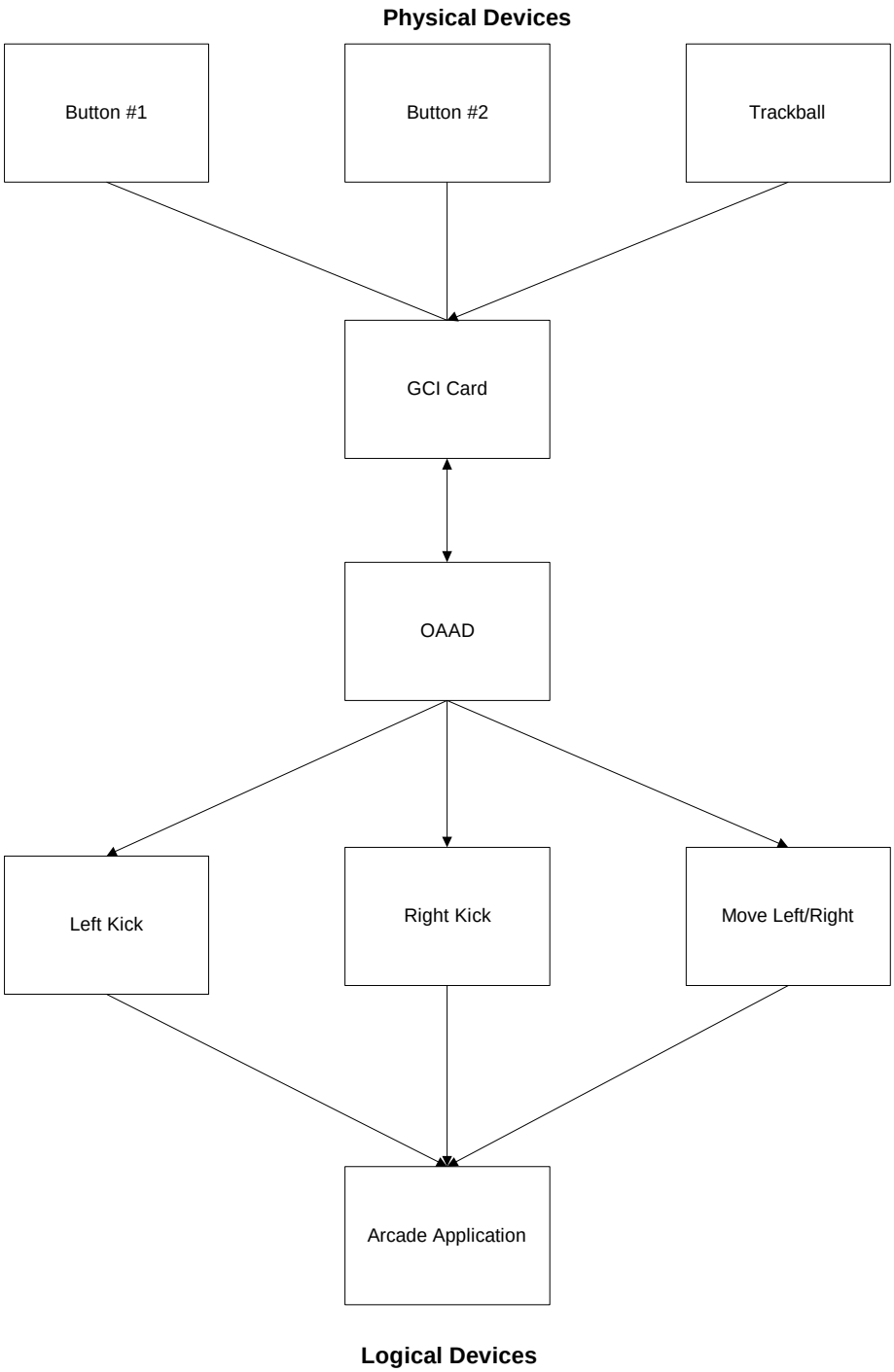
The OAADDevice object is intended to shield the arcade application developer from having to deal with the physical devices directly. This means that the OAAD Library has to provide support for physical devices in a logical and consistent manner. There is an OAADDevice object for each physical OAAD device connected to the system.

3 Mapping of Logical to Physical devices

Since each system may be wired differently, and different hardware interface cards may offer different types of input and output the developer should be able to write applications in a standardized manner. For example, instead of reading input from port 'ppp', and masking bit 'n' to determine that the Right Kick Button has been pressed, the developer should be able to Request the status of the 'Right Kick Button' and OAAD handles the mapping of this logical device to the physical one.

Since many of the devices do not provide any mechanism to identify themselves a mapping mechanism will be provided by OAAD that allows the installer of the game to map each physical device to the desired logical device. This mapping will be configured by discovery. For example, the installer could be prompted to press the 'Right Kick Button' and the mapping mechanism will record the appropriate relationship between the physical device and the logical representation that the arcade application will be using to access it. At the present time this mapping mechanism is still under discussion, and will be finalized in the next revision of this document.

The following diagram illustrates a simple example of physical and logical devices.



3.1.1 Polling and Events

There are two methods that may be used to determine if input data is available:

- Polling
- Event notification.

Polling a device means requesting the current state of the device objects with **IOAADDevice::GetDeviceState**, or retrieving the contents of any buffered data with **IOAADDevice::GetDeviceData**. Polling is typically only used for data and state information that is expected to be present for a long enough time that it will still be available when the next poll request is made. In the OAADDevice object event data is queued in conjunction with the Game Controller Interface card hardware. This reduces the likelihood of event loss if there is some delay before the application can request the data or state from the OAADDevice object.

For new arcade application development WIN32 event notification should be used for all cases where it is critical that the application does not miss any information. For example, counting pulses from coin doors. To use event notification, the developer sets up a thread synchronization object using the Win32 **CreateEvent** function and then associates this event with the target device by passing the handle to the event object to the **IOAADDevice::SetEventNotification** method. The event is then signaled by OAAD whenever the state of the device changes. The arcade application can receive notification of the event by using one of the Win32 functions, such as **WaitForSingleObject**, then respond by checking the input buffer to find out what the event was. For sample code, see the ***'To be written'*** sample and the reference for **IOAADDevice::SetEventNotification**.

OAAD Device Driver Developers should provide support for event notification in their OAAD implementations. This support can be used in conjunction with the built in event queuing.

Many arcade applications are implemented using a "pipeline", rather than a multi-threaded, architecture. For these single threaded legacy game implementations, the OAADDevice object's implementation of queuing events in conjunction with the Game Controller Interface card should be used to ensure that events are not lost. In this case the game needs to call the OAADDevice object GetDeviceState and/or GetDeviceData methods in the pipeline often enough to ensure that the hardware queue does not overflow.

3.1.2 Watchdog Timers

OAAD provides support for Watchdog timers in the same fashion as other devices. Watchdog timers are used to allow recovery from a locked up application or computer. An interval is set and the timer begins counting down. An action is specified that will take place if the timer reaches 0. It is up to the arcade application to reset the timer before it reaches 0. The data that is associated with the Watchdog timer consists of two parameters:

dwTimeInterval	Specifies the number of seconds before the Watchdog timer times out. If set to 0 then the Watchdog timer is disabled.
dwAction	The action that is to be associated with the expiration of the Watchdog timer interval. Currently only two actions are defined: OAWDT_REBOOT Causes the computer running the application to initiate a hardware re-boot. OAWDT_RESTART Causes the arcade application to be restarted, without a hardware reboot of the computer.

The action that is taken when the Watchdog timer expires is dependent on the interface card. For example, if the card does not provide a physical connection for pulling the computer's reset line low then OAWDT_REBOOT will not have any effect. Also note that even if the interface card supports a reset

capability this may not be what should be done, especially if the system is running the Windows NT OS which should perform an orderly shutdown.

3.2 Functions

Devices that are already fully supported by the DirectInput API do not require any additional support from OAAD. OAAD provides support for those devices that are not supported, or only partially supported, by DirectInput. Wherever possible this support is provided by DirectInput-like methods, which have been extended to support generalized output and multiple data reports per device.

DirectInput (up through version 6, mid-1998) assumed that any device creates only one "record" or set of data items. The **GetDeviceData**/**GetDeviceState** methods return one block of data, up to 32-bits-long. And each device has only one DIDATAFORMAT structure describing the data, which DirectInput uses to arrange data as the app requests via a **SetDataFormat** call. However, some arcade devices, such as a bill acceptor, may want to return more than one datum or multiple logical groupings specified by the application. The OAAD SDK provides mechanisms for dealing with these multiple datums by constructing OADATAFORMAT structures from HID information, possibly more than one per device.

3.2.1 Device Function Calls

The devices shown in Table 1, commonly found in a coin-operated machines, will be supported, except as noted. Support consists of essentially 6 types or categories:

- Querying/reporting device existence and capabilities
- Configuring device addresses and data reporting
- Reading input status
- Reading input data (registers)
- Writing commands to devices
- Writing data and/or data-levels to devices

Devices such as joysticks, trackballs, keyboards, mice, and force-feedback joysticks already supported by the DirectInput API need no additional support in OAAD, and thus are not mentioned in the list. Other Windows APIs provide support for other devices, such as cameras, printers, etc. Most of the arcade devices are supported by DirectInput-like calls. These calls have been extended to support generalized output and multiple data records per device.

Table 1: Coin-op Devices

Device	**Static Queries (Properties)**	**Input and/or Status Calls**	**Output Calls**
Back-lit buttons (toggling)		Up/down	On, off, toggle
Bill acceptor		Status (bill-inserted, reject, full, jam), count	
Bill lockout output (disables bill acceptor from taking bills)		Locked/unlocked	Lockout/unlock
Coin acceptor (Coin mechanism)		Open/closed, Full, coinage received	Reset, Unlock

Coin lockout output (disables coin acceptor from taking coins)		Locked/unlocked	Lockout/unlock
Coin meter outputs	#digits	Get count	Reset, Increment, Decrement
Coin hopper output		Get count	Dispense, count
Coupon (ticket) printers		Status (empty, jam), count	Print, amount, string field
Credit/Debit card swipe readers		Status(inserted, reject, jammed), card ID	
Game panel Lights		N/A	On/off
Key-locks		Locked/unlocked	Lock/unlock
Light array matrixes			
Motion Detector (infrared, ala Interactive Light)			
Light pen or gun***		Count (or X, Y position?)	Reset counter
Motion Detectors	Range	Count (position,range?)	
Motion chairs (E.G, Jessler, RockNRide*, VirToGo*),	# Degrees of Motion	Speed, acceleration (if supports friction).	Degree of freedom, motion speed (angular or linear), acceleration (angular or linear), friction.
PIN-pads (Alphanumeric keypad for entering Personal ID #) (??)		ID, Enter, Correction, Cancel	Display message, Accept and Encrypt PIN
Reel mechanisms (like slot machines)		State (apple, cherry, etc), spinning/stopped, count	Spin
Rotary beacon lights		Enabled/disabled	Enable/disable
Ticket meter outputs	(same as coin meters)		
Ticket dispensers		Status (empty, jam), count	Dispense, amount, string field.
Touch-screens			
Vending Dispenser Motors		Dispensed, Jam/Error, Empty, Count Remaining	Dispense
Watchdog timers	Size, interval	Count, Start/Stop	Reset, Load

**Gray-shaded items are of secondary priority, and may not be supported in the initial release of the OAAD SDK. Items marked with double-question-marks '??' may remain unsupported in future releases. The need and difficulty of supporting devices designated with '??' remains uncertain.

*** Light guns, and pens, require the display screen to be driven brighter (preferably white), during the retrace periods. This implies display driver or DirectDraw* involvement.

Security dongles are not supported by OAAD. The reason for this is that, while dongles are frequently used in arcade machines, the application software does not directly access them. The dongle support software is typically a post-processing step that wraps the application software.

As in DirectInput, data for input and/or output must be "Buffered" or "Immediate". Furthermore, most devices must support both polled and event-driven I/O. Other devices with different functionality will need support in the future, so the software must be easily extensible to new devices, preferably via the Windows Registry or a Windows .INI file.

All devices are assumed connected to the PC either via the game-port, serial communications port, Universal Serial Bus, parallel port, or an interface card such as the Quantum3D GCI-2* (which itself connects to a PC serial communications port).

3.2.2 Generic Functions

Watch dog timer support is provided under OAAD, if the underlying hardware supports it. The parameters for the timer are configurable by the application, as well as the action to be taken if a watch dog timer fires. One action could be to have the interface card re-boot the PC if no activity is detected for a period of time. Note: Hardware re-boot requires that the interface card be able to control the hardware-reset line on the computer's bus.

The following OAAD (Open Arcade Architecture Device) methods are currently defined:

HRESULT WINAPI OAADCreate

TheOAADCreate function creates an instance of an OAAD object that supports the IOAAD COM interface. Note: This is a 'C' callable function.

```
HRESULT WINAPI OAADCreate(
  HINSTANCE hInstance,
  DWORD dwVersion,
  LPOAAD * lplpOAAD,
  LPUNKNOWN punkOuter
);
```

Parameters

hInstance
Instance handle to the application or DLL that is creating the OAAD object. OAAD uses this value to determine whether the application or DLL has been certified and to establish any special behaviors that may be necessary for backward compatibility. May be set to NULL.

It is an error for a DLL to pass the handle of the parent application. For example, an ActiveX control embedded in a container that uses OAAD must pass its own instance handle and not the handle of the container application. This ensures that OAAD recognizes the control and can enable any special behaviors that may be necessary. Note that if this parameter is set to NULL then any special actions that might be required by the container application will not be enabled.

dwVersion
This parameter contains the version number of the OAAD library, which the application requires. This value will normally be OAAD_VERSION. Passing the version number of a previous version will cause OAAD to emulate the functionality of that version.

lplpOAAD
Address of a variable to receive a valid IOAAD interface pointer if the call succeeds.

punkOuter
Contains the Pointer to the address of the controlling object's IUnknown interface for COM aggregation or NULL if the interface is not aggregated. Most callers will pass NULL. If aggregation is requested, the

object returned in *lplpOAAD will be a pointer to the IUnknown rather than an IOAAD interface, as required by the COM aggregation specification.

Return Values

If the function succeeds, the return value is OA_OK.
If the function fails, the return value may be one of the following error values:

OAERR_BETAOAADVERSION
OAERR_INVALIDPARAM
OAERR_OLDOAADVERSION
OAERR_OUTOFMEMORY

Remarks

Calling this function with punkOuter = NULL is equivalent to creating the object through CoCreateInstance(&CLSID_OAAD, punkOuter, CLSCTX_INPROC_SERVER, &IID_IOAAD, lplpOAAD), then initializing it with Initialize.

Calling this function with punkOuter != NULL is equivalent to creating the object through CoCreateInstance(&CLSID_OAAD, punkOuter, CLSCTX_INPROC_SERVER, &IID_IUnknown, lplpOAAD). The aggregated object must be initialized manually.

There are separate ANSI and Unicode versions of this service. The ANSI version creates an object that supports the IOAADA interface, whereas the Unicode version creates an object that supports the IOAADW interface. As with other system services that are sensitive to character set issues, macros in the header file map OAADCreate to the appropriate character set variation.

OAAD::CreateDevice

CreateDevice is used to create an instance of the requested OAAD device.

```
HRESULT CreateDevice(
  REFGUID rGUID,
  LPOAADDEVICE *lplpOAADDevice,
  LPUNKNOWN pUnkOuter
);
```

Parameters

rGUID
Reference to (C++) or address of (C) the instance GUID for the desired input device (see Remarks). The GUID is retrieved through the IOAAD::EnumDevices method, or it can be one of the following predefined GUIDs:

GUID_???? TBD.
GUID_???? TBD.
For the above GUID values to be valid, your application must define INITGUID before all other preprocessor directives at the beginning of the source file, or link to OAADguid.lib.

lplpOAADDevice
Address of a variable to receive the IOAADDevice interface pointer if successful.

pUnkOuter
Address of the controlling object's IUnknown interface for COM aggregation, or NULL if the interface is not aggregated. Most callers will pass NULL.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following:

OAERR_DEVICENOTREG
OAERR_INVALIDPARAM
OAERR_NOINTERFACE
OAERR_NOTINITIALIZED
OAERR_OUTOFMEMORY

Remarks

In C++ the rguid parameter must be passed by reference; in C, which does not have pass-by-reference, it must be passed by address. The following is an example of a C++ call:

lpOAAD->CreateDevice(GUID_????, &pdev, NULL);

The following shows the same call in C:

lpOAAD ->lpVtbl->CreateDevice(lpOAAD, &GUID_????, &pdev, NULL);

Calling this function with punkOuter = NULL is equivalent to creating the object via CoCreateInstance(&CLSID_OAADDevice, NULL, CLSCTX_INPROC_SERVER, riid, lplpOAADDevice) and then initializing it with Initialize.
Calling this function with punkOuter != NULL is equivalent to creating the object via CoCreateInstance(&CLSID_OAADDevice, punkOuter, CLSCTX_INPROC_SERVER, &IID_IUnknown, lplpOAADDevice). The aggregated object must be initialized manually.

Note: The GUID that is passed in will normally have been established by enumerating the devices in the system, via the EnumDevices method.

OAAD::EnumDevices

The EnumDevice method enumerates devices that are either currently attached or could be attached to the computer.

```
HRESULT EnumDevices(
  DWORD dwDevType,
  LPOAADENUMCALLBACK lpCallback,
  LPVOID pvRef,
  DWORD dwFlags
);
```

Parameters

dwDevType
Device type filter. If this parameter is zero, all device types are enumerated. Otherwise, it is a OADEVTYPE_* value (see OADEVICEINSTANCE), indicating the device type that should be enumerated.

lpCallback
Address of a callback function that will be called with a description of each OAAD device.

pvRef
An application-defined 32-bit value that will be passed to the enumeration callback each time it is called.

dwFlags
Flag value that specifies the scope of the enumeration. This parameter can be one or more of the following values.

OAEDFL_ALLDEVICES

All installed devices will be enumerated.

OAEDFL_ATTACHEDONLY

Only attached and installed devices. This is the default behavior.

OAEDFL_COINACCEPTOR

Only coin acceptor type devices will be enumerated.

Others TBD.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following error values:

OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED

Remarks

Keep in mind that all installed devices can be enumerated, even if they are not currently connected to the computer. For example, a flight stick may be installed on the system but not currently plugged into the computer. The dwFlags parameter may be set to indicate whether only attached, or all installed devices, should be enumerated. If the OAEDFL_ATTACHEDONLY flag is not present, all installed devices will be enumerated. A preferred device type can be passed as a dwDevType filter so that only the devices of that type are enumerated, E.G. coin acceptors.

The lpCallback parameter specifies the address of a callback function of the type documented as OAADEnumDevicesProc. OAAD calls this function for every device that is enumerated. In the callback, the device type and friendly name, and the product GUID and friendly name, are given for each device.

OAAD::GetVersion

The GetVersion method returns the release version of the installed OAAD SDK, or an error codes if the installation was unsuccessfully attempted.

HRESULT GetVersion(LPDWORD pdwVersion);

Parameters

pdwVersion
Contains the pointer to a DWORD that will receive the version number of the currently installed OAAD SDK. This version number will be in the same format as the DirectInput version number. EG, 5.0 will be represented as 0x0500. Refer to the OAAD oaad.h header file for an example.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following error values:

OA_NOTINITIALIZED
OA_NOTINSTALLED

OAAD::GetDeviceStatus

Tells whether, or not, the device is attached to the system and initialized.

```
HRESULT GetDeviceStatus(
  REFGUID rguidInstance
);
```

Parameters

rguidInstance
Instance GUID identifier of the device whose status is being checked.

Return Values

If the method succeeds, the return value is OA_OK if the device is attached to the system, or OA_NOTATTACHED otherwise. If the method fails, the return value may be one of the following error values:

OAERR_GENERIC
OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED

OAADDevice::GetDeviceInfo

This method obtains information about the device's identity into an OADeviceInstance structure.

```
HRESULT GetDeviceInfo(
  LPOADEVICEINSTANCE pOAADdi
);
```

Parameters

pOAADdi
Address of an OADEVICEINSTANCE structure to be filled with information about the device's identity. The application must be certain to initialize the structure's dwSize member before calling this method.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following error values:

OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED

OAADDevice::SetEventNotification

Associates an event (Created with the Win32 CreateEvent function) with a particular device. This is used for event driven input

```
HRESULT SetEventNotification(
  HANDLE hEvent
);
```

Parameters

hEvent
Handle to the event that is to be set when the Device State changes. OAAD will use the Win32 SetEvent function on the handle when the state of the device changes. If the hEvent parameter is NULL, then notification is disabled.

The application may create the handle as either a manual-reset or automatic-reset event by using the Win32 CreateEvent function. If the event is created as a automatic-reset event, then the operating system will automatically reset the event once a wait has been satisfied. If the event is created as a manual-reset event, then it is the application's responsibility to call the Win32 ResetEvent function to reset it. OAAD will not call the Win32 ResetEvent function for event notification handles. Most applications will create the event as an automatic-reset event.

Return Values

If the method succeeds, the return value is OA_OK or OA_POLLEDDEVICE. If the method fails, the return value may be one of the following error values:

OAERR_ACQUIRED
OAERR_HANDLEEXISTS
OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED

Remarks

A device state change is defined as any of the following:

- A change in the position of an axis
- A change in the state (pressed or released) of a button
- A change in the direction of a POV control
- Loss of acquisition
- A coin acceptor pulse

Do not call the Win32 CloseHandle function on the event while it has been selected into an OAADDevice object. You must call this method with the hEvent parameter set to NULL before closing the event handle. The event notification handle cannot be changed while the device is acquired. If the function is successful, then the application can use the event handle in the same manner as any other Win32 event handle.

OAADDevice::SetEventNotificationEx

Associates an event (Created with the Win32 CreateEvent function) with a particular data report on a device. This is used for event driven input, similar to SetEventNotification but it directly calls back to the application passing the data associated with the registered event.

```
HRESULT SetEventNotificationEx(
  LPOAADEVNTCALLBACK pCallback,
  DWORD dwUsageID,
  HANDLE hEvent
);
```

Parameters

pCallback
Pointer to an OAADEVNTCALLBACK function that will process the associated data. This parameter may be NULL. If it is NULL then the device object will not be able to return any data to the application. If this parameter is NULL then the hEvent parameter should also be set to NULL.

dwUsageID
The Usage ID for the associated data that the event is associated with.

hEvent
Handle to the event that is to be signaled when the Device State changes. OAAD will use the Win32 SetEvent function on the handle when the state of the device changes. If the hEvent parameter is NULL, then notification is disabled. If this parameter is NULL then the pCallback parameter should also be set to NULL.

The application may create the handle as either a manual-reset or automatic-reset event by using the Win32 CreateEvent function. If the event is created as a automatic-reset event, then the operating system will automatically reset the event once a wait has been satisfied. If the event is created as a manual-reset event, then it is the application's responsibility to call the Win32 ResetEvent function to reset it. OAAD will not call the Win32 ResetEvent function for event notification handles. Most applications will create an automatic-reset event.

Return Values

If the method succeeds, the return value is OA_OK or OA_POLLEDDEVICE. If the method fails, the return value may be one of the following error values:

OAERR_ACQUIRED
OAERR_HANDLEEXISTS
OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED

Remarks

A device state change is defined as any of the following:

- A change in the position of an axis
- A change in the state (pressed or released) of a button
- A change in the direction of a POV control
- Loss of acquisition
- A coin acceptor pulse

Do not call the Win32 CloseHandle function on any of the events that have been registered while they are selected into an OAADDevice object. You must call this method with the hEvent and pCallback parameters set to NULL before closing the event handle. Or you may call the ReleaseAllEventNotifications method before closing the event handles.

An event notification handle cannot be changed while the device is acquired. If the function is successful, then the application can use the event handle in the same manner as any other Win32 event handle.

When the data associated with the specified UsageID changes the callback function will be called by the device object. A pointer to the associated data will be passed, along with the size of the data in the device object. There is an additional parameter that will be NULL if this is a final call on shutdown of the device object. The application should minimize the time that it spends processing the data in the callback function to maintain performance. See the GUIExample application for more details.

OAADDevice::ReleaseAllEventNotifications

The method releases all of the extended event notification handles that were set by previous calls to SetEventNotificationEx. This should be called when the application is finished with data from the device. No more callbacks will be made by the device object after this call returns.

Parameters

None.

Return Values

OA_OK or error code.

OAADDevice::EnumObjects

Lists the objects (buttons, etc.) that are available on a given device. In HID-terminology, this method returns a list of all of the usages that are supported by the device.

```
HRESULT EnumObjects(
  LPOAENUMDEVICEOBJECTSCALLBACK lpCallback,
  LPVOID pvRef,
  DWORD dwFlags
);
```

Parameters

lpCallback
Address of a callback function that receives OAADDevice objects. OAAD provides a prototype of this function as OAADEnumDeviceObjectsProc. This callback function is passed a pointer to an OADEVICEOBJECTINSTANCE structure that contains the information describing the object being enumerated.

pvRef
An application-defined 32-bit value that will be passed to the enumeration callback each time it is called.

dwFlags
Contains Flags specifying the types of object to be enumerated. Each of the following values restricts the enumeration to objects of the described type:

OADFT_ALL

All objects.

OADFT_BUTTON

A push button or a toggle button.

OADFT_COLLECTION

A HID link collection. HID link collections do not generate data of their own.

OADFT_ENUMCOLLECTION(n)

An object that belongs to HID link collection number n.

OADFT_NOCOLLECTION

An object that does not belong to any HID link collection. In other words, an object for which the wCollectionNumber member of the OADEVICEOBJECTINSTANCE structure is set to 0.

OADFT_NODATA

Defines an object that does not generate any data.

OADFT_OUTPUT

Defines an object to which data can be sent by using the SendDeviceData method.

OADFT_PSHBUTTON

A push button. A push button is reported as down when the user presses it and as up when the user releases it.

OADFT_TGLBUTTON

A toggle button. A toggle button is reported as down when the user presses it and remains so until the user presses the button a second time.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following error values:

OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED

OAADDevice::GetObjectInfo

This method retrieves information about a particular device object, E.G, a button.

```
HRESULT GetObjectInfo(
  LPOADEVICEOBJECTINSTANCE pdidoi,
  DWORD dwObj,
  DWORD dwHow
);
```

Parameters

pdidoi

Address of a OADEVICEOBJECTINSTANCE structure to be filled with information about the object. The structure's dwSize member must be initialized before this method is called.

dwObj

Value that identifies the object whose information will be retrieved. The value set for this parameter depends on the value specified in the dwHow parameter.

dwHow

Value specifying how the dwObj parameter should be interpreted. This value can be one of the following:

Value	**Meaning**
OAPH_DEVICE	The dwObj parameter must be zero.
OAPH_BYOFFSET	The dwObj parameter is the offset into the current data format of the object whose information is being accessed.
OAPH_BYID	The dwObj parameter is the object type/instance identifier. This identifier is returned in the dwType member of the OADEVICEOBJECTINSTANCE

	structure returned from a previous call to the IOAADDevice::EnumObjects method.
OAPH_BYUSAGE	The dwObj parameter contains the HID Usage Page and Usage values of the object, combined by the OAMAKEUSAGEDWORD macro.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following error values:

OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED
OAERR_OBJECTNOTFOUND

OAADDevice::EnumDataFormats

The EnumDataFormats method enumerates the data formats that are associated with a device. Note that there may be more than one data format associated with a device.

```
HRESULT EnumDataFormats (
  DWORD dwReportType,
  DWORD dwReportID,
  LPOAADENUMCALLBACK lpCallback,
  LPVOID pvRef,
  DWORD dwFlags
);
```

Parameters

dwReportType
Report type filter. If this parameter is zero, all report types are enumerated. Otherwise, it is an OART_* value (see OADATAINFO), indicating the report type that should be enumerated.

dwReportID
Contains a Report ID filter. If this parameter is zero, all report ID are enumerated. Otherwise, it is a value indicating the report ID that should be enumerated.

lpCallback
Address of a callback function that will be called with a description of each Report Format.

pvRef
An application-defined 32-bit value that will be passed to the enumeration callback each time it is called.

dwFlags
Flags allow an application to force the data format to be enumerated. The value may be one or more of the following:

OADFT_NONE
Does not use any predefined OADATAFORMAT structures.

TBD.

Return Values

If the method succeeds, the return value is OA_OK.

If the method fails, the return value may be one of the following error values:
OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED

Remarks

A preferred report type can be passed as a dwReportType filter so that only the reports of that type are enumerated.

The lpCallback parameter specifies the address of a callback function of the type documented as OAEnumFormatsProc. OAAD calls this function for every data format that is enumerated. In the callback,

the report type and report ID, are given for each data format. If a single device can generate more than one report type, it will be returned for each report type it supports. For example, a device that declares a Feature and an Input report types, both will be enumerated.

OAADDevice::EnumDataFormatsProc

The EnumDataFormatsProc function is an application-defined callback function that receives OAAF Data Formats as a result of a call to the EnumDataFormats method.

```
BOOL CALLBACK EnumDataFormatsProc(
  LPCOADATAINFO lppodi,
  LPVOID pvRef
);
```

Parameters

lpddoi
An OADATAINFO structure that describes the data report being enumerated.

The report number for the current data format may be accessed as lppodi->dwReportID. This member contains the Report Usage ID number associated with the data format and may be used to equate the data format with a particular Usage ID.

pvRef
Contains the application-defined value given in the EnumDataFormats method.

Return Values

Returns OAENUM_CONTINUE to continue the enumeration or OAENUM_STOP to stop the enumeration.

OAADDevice::GetDeviceState

The GetDeviceState method retrieves instantaneous data from the device. An application may call this method at any time, and will receive the current state/data from the device. If the state/data has not changed since a previous call to this method that previous state will be returned, if appropriate. E.g. it would not be correct to return a 'coin dropped' status each time this method was called, but it would be OK to return that a ticket printer was still out of paper.

HRESULT GetDeviceState(
DWORD cbData,
LPVOID lpvData,
LPOADATAINFO lpodi
);

Parameters

cbData
Size of the buffer in the lpvData parameter, in bytes.

lpvData
Address of a structure that receives the current state of the device. The format of the data identified by lpoid.

lpodi
Points to an OADATAINFO structure that identifies which OADATAFORMAT structure will be returned. Only the OART_FEATURE report type can be used when retrieving instantaneous data from a device.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following error values:

OAERR_INPUTLOST
OAERR_INVALIDPARAM
OAERR_NOTACQUIRED
OAERR_NOTINITIALIZED
E_PENDING

Remarks

Before device data can be obtained, set the cooperative level by using the IOAADDevice::SetCooperativeLevel method, then set the data format by using IOAADDevice::SetDataFormat, and acquire the device by using the IOAADDevice::Acquire method.

OAADDevice::SetDeviceState

The SetDeviceState method sends instantaneous data to the device. The specific report type and ID are indicated by the OADATAINFO structure passed by the application.

HRESULT SetDeviceState(
DWORD cbData,
LPVOID lpvData,
LPOADATAINFO lpodi
);

Parameters

cbData
Size of the buffer in the lpvData parameter, in bytes.

lpvData
Address of a structure that contains the new state of the device. The format of the data is established by a prior call to the GetDataFormat method.

lpodi
Points to an OADATAINFO structure that identifies the OADATAFORMAT structure that will be sent. Only the OART_FEATURE and the OART_OUTPUT report types can be used when sending instantaneous data to a device.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following error values:

OAERR_INPUTLOST
OAERR_INVALIDPARAM
OAERR_NOTACQUIRED
OAERR_NOTINITIALIZED
OA_DEVICEBUSY

Remarks

OAADDevice::SetDeviceData

The SetDeviceData method is new to OAAD. This method is used to send low latency, buffered data to the device.

HRESULT SetDeviceData(
DWORD cbObjectData,
LPOADEVICEOBJECTDATA rgdod,
LPDWORD pdwInOut,
LPOADATAINFO lpodi
);

Parameters

cbObjectData
Size of the OADEVICEOBJECTDATA structure, in bytes.

rgdod
Array of OADEVICEOBJECTDATA structures that contain the buffered data. The number of elements in this array must be equal to the value of the pdwInOut parameter.

pdwInOut
On entry, the number of elements in the array pointed to by the rgdod parameter. On exit, the number of elements actually set.

lpodi
Points to an OADATAINFO structure that identifies the report type, ID and OADATAFORMAT of the LPOADEVICEOBJECTDATA structure is being sent. The OART_OUTPUT Report Type can be used when sending buffered data to a device.

Returns

If the method succeeds, the return value is OA_OK or OA_BUFFEROVERFLOW.
If the method fails, the return value may be one of the following error values:

OAERR_INVALIDPARAM
OAERR_NOTACQUIRED
OAERR_NOTBUFFERED
OAERR_NOTINITIALIZED

Remarks

Before device data can be obtained, you must set the data format by using the IOAADDevice::SetDataFormat method, set the buffer size with IOAADDevice::SetProperty method, and acquire the device by using the IOAADDevice::Acquire method.

Buffered data is written to the device over the Output pipe (or the Control pipe if the device does not declare an Output pipe).

OAADDevice::GetDeviceData

The GetDeviceData method retrieves buffered data from the Input pipe of a HID device. An Input pipe can return any of the input reports defined by a device. The OADATAINFO structure will identify the specific report returned.

HRESULT GetDeviceData(
DWORD cbObjectData,
LPOADEVICEOBJECTDATA rgdod,
LPDWORD pdwInOut,
DWORD dwFlags,
LPOADATAINFO *lpodi
);

Parameters

cbObjectData

Size of the OADEVICEOBJECTDATA structure, in bytes. This should be large enough to hold the largest OART_INPUT report. In the DEVICEOBJECTDATA the dwSequence holds a value that represents the Usage ID and Report ID. The Usage ID is held in the high order WORD and the Report ID is in the low order WORD.

rgdod

Array of OADEVICEOBJECTDATA structures to receive the buffered data. The number of elements in this array must be equal to the value of the pdwInOut parameter. If this parameter is NULL, then the buffered data is not stored anywhere, but all other side-effects take place.

pdwInOut

On entry, the number of elements in the array pointed to by the rgdod parameter. On exit, the number of elements actually obtained.

dwFlags

Contains Flags that control the manner in which data is obtained. This value may be zero or the following flag.

0	The data entry will be removed from the queue.
OAGDD_PEEK	Does not remove the data entry from the queue. A subsequent GetDeviceData method call will read the same data. Normally, data is removed from the buffer after it is read.

lpodi

GetDeviceData can return any report ID of OART_INPUT report type. lpodi points to an array of pointers to OADATAINFO structures that identifies the report IDs and OADATAFORMATs of the returned data. The last pointer in the list will be NULL and an application should test for the NULL pointer to determine the end of the list. This information should already be available to the application from calls to the EnumObjects and EnumDataFormats methods. The array that is returned is managed by the OAAD Device Object, the application should NOT delete this memory. This parameter may be NULL.

Return Values

If the method succeeds, the return value is OA_OK or OA_BUFFEROVERFLOW if the Queue has overflowed and entries have been lost.

If the method fails, the return value may be one of the following error values:

OAERR_INPUTLOST
OAERR_INVALIDPARAM
OAERR_NOTACQUIRED
OAERR_NOTBUFFERED
OAERR_NOTINITIALIZED

Remarks

Before device data can be obtained, you must set the data format by using the IOAADDevice::SetDataFormat method, set the buffer size with IOAADDevice::SetProperty method, and acquire the device by using the IOAADDevice::Acquire method.

OAADDevice::GetProperty

The GetProperty method checks the rguidProp parameter for the OAAD specific properties and returns the appropriate information from the device: EG, the number of OADATAFORMAT structures provided by a device. This overrides DirectInput's **GetProperty** method.

HRESULT GetProperty(
REFGUID rguidProp,
LPOAPROPHEADER pdiph
);

Parameters

rguidProp
Contains the Identifier of the property to be retrieved. This can be one of the predefined values, or a pointer to a GUID that identifies the property. The following properties are predefined for an input device.

OA_DATAFORMAT_STRUCTS

Specifies the number of OADATAFORMAT structures provided by the device.

OAPROP_BUFFERSIZE

Retrieves the input-buffer size. The retrieved value is set in the dwData member of the associated OAADPROPDWORD structure. See the description for the pdiph parameter for more information.
The buffer size determines the amount of data that the buffer can hold between calls to the IOAADDevice::GetDeviceData method before data is lost. This value may be set to zero to indicate that the application will not be reading buffered data from the device. If the buffer size in the dwData member of the OAPROPDWORD structure is too large to be supported by the device, the largest possible buffer size is set. To determine whether

the requested buffer size was set, retrieve the buffer-size property and compare the result with the value you previously attempted to set.

OAPROP_FFLOAD

Retrieves the memory load for the device. This setting applies to the entire device, rather than to any particular object, so the dwHow member of the associated OAPROPDWORD structure must be OAPH_DEVICE.
The dwData member contains a value in the range 0 to 100, indicating the percentage of device memory in use.

OAPROP_GUIDANDPATH

Allows the application to access the class GUID and device interface (path) for the device. This property lets advanced applications perform operations on the device that are not supported by OAAD. For more information, see the reference for the OAPROPGUIDANDPATH structure.

OAPROP_RANGE

Retrieves the range of logical values an object can possibly report. The retrieved minimum and maximum values are set in the lMin and lMax members of the associated OAPROPRANGE structure. See the description for the poaph parameter for more information.
For some devices, this is a read-only property; you cannot set its value by calling the IOAADDevice::SetProperty method.

OAPROP_PHYSICAL_EXTENTS

Retrieves the range of physical values an object can possibly report. The retrieved minimum and maximum values are set in the lMin and lMax members of the associated OAPROPRANGE structure. See the description for the poaph parameter for more information.
For some devices, this is a read-only property; you cannot set its value by calling the IOAADDevice::SetProperty method.

OAPROP_REPORTID

Retrieves the HID Usage Report ID associated with the object. This is a read-only property.

OAPROP_FLAGS

Main item flags, Constant, Data, Array, etc.

pdiph
Address of the OAPROPHEADER portion of a larger property-dependent structure that contains the OAPROPHEADER structure as a member. When retrieving object range information, this value is the address of the OAPROPHEADER structure contained within the OAPROPRANGE structure. For most other properties, this value is the address of the OAPROPHEADER structure contained within the OAPROPDWORD structure.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following error values:

OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED
OAERR_OBJECTNOTFOUND
OAERR_UNSUPPORTED

Remarks

Since this method is now an override of an existing DI function the definitions for the rguidProp are being left in. This may be changed in a future revision of this document.

OAADDevice::SetProperty

The SetProperty method is the complement of the GetProperty method.

```
HRESULT SetProperty(
  REFGUID rguidProp,
  LPCOAPROPHEADER pdiph
);
```

Parameters

rguidProp

Contains the Identifier of the property to be set. This can be one of the predefined values, or a pointer to a GUID that identifies the property. The following property values are predefined for an input device.

OAPROP_BUFFERSIZE

Sets the input-buffer size. The value being set must be specified in the dwData member of the associated OAPROPDWORD structure. See the description for the pdiph parameter for more information.
This setting applies to the entire device, so the dwHow member of the associated OAPROPDWORD structure must be set to OAPH_DEVICE.

OAPROP_RANGE

Sets the range of values an object can possibly report. The minimum and maximum values are taken from the lMin and lMax members of the associated OAPROPRANGE structure.
For some devices, this is a read-only property. You cannot set a reverse range; lMax must be greater than lMin.

pdiph

Address of the OAPROPHEADER structure contained within the OAPROPDWORD structure. If setting object range information, this is the address of the OAPROPHEADER structure contained within the OAPROPRANGE structure.

Return Values

If the method succeeds, the return value is OA_OK or OA_PROPNOEFFECT.

If the method fails, the return value may be one of the following error values:

OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED
OAERR_OBJECTNOTFOUND
OAERR_UNSUPPORTED

Remarks

The buffer size determines the amount of data that the buffer can hold between calls to the IOAADDevice::GetDeviceData method before data is lost. This value may be set to zero to indicate that the application will not be reading buffered data from the device. If the buffer size in the dwData member of the OAPROPDWORD structure is too large to be supported by the device, the largest possible buffer size is set. To determine whether the requested buffer size was set, retrieve the buffer-size property and compare the result with the value you previously attempted to set.

OAADDevice::SetCooperativeLevel

The SetCooperativeLevel method establishes the cooperative level for this instance of the device. The cooperative level determines how this instance of the device interacts with other instances of the device and the rest of the system.

```
HRESULT SetCooperativeLevel(
  HWND hwnd,
  DWORD dwFlags
);
```

Parameters
hwnd

Contains the Window Handle to be associated with the device. This parameter must be a valid top-level window handle that belongs to the process. The window associated with the device must not be destroyed while it is still active in a DirectInput device.

dwFlags

Flags that describe the cooperative level associated with the device. This parameter can be one of the following values:

OASCL_BACKGROUND

The application requires background access. If background access is granted, the device may be acquired at any time, even when the associated window is not the active window.

OASCL_EXCLUSIVE

The application requires exclusive access. If exclusive access is granted, no other instance of the device may obtain exclusive access to the device while it is acquired. Note, however, non-exclusive access to the device is always permitted, even if another application has obtained exclusive access.
An application that acquires the mouse or keyboard device in exclusive mode should always un-acquire the devices when it receives WM_ENTERSIZEMOVE and WM_ENTERMENULOOP messages. Otherwise, the user will not be able to manipulate the menu or move and resize the window.

OASCL_FOREGROUND

The application requires foreground access. If foreground access is granted, the device is automatically un-acquired when the associated window moves to the background.

OASCL_NONEXCLUSIVE

The application requires non-exclusive access. Access to the device will not interfere with other applications that are accessing the same device.

Applications must specify either OASCL_FOREGROUND or OASCL_BACKGROUND; it is an error to specify both or neither. Similarly, applications must specify either OASCL_EXCLUSIVE or OASCL_NONEXCLUSIVE.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following error values:

OAERR_INVALIDPARAM
OAERR_NOTINITIALIZED

Remarks

Applications must call this method before acquiring the device by using the IOAADDevice::Acquire method.

OAADDevice::Escape

A generic 32 bit output, device specific, driver specific sequence.

```
HRESULT Escape(
  LPOAEFFESCAPE pesc
);
```

Parameters
pesc

Address of an OAEFFESCAPE structure that describes the command to be sent. On success, the cbOutBuffer member contains the number of bytes of the output buffer actually used.

Return Values

If the method succeeds, the return value is OA_OK.
If the method fails, the return value may be one of the following error values:

OAERR_DEVICEFULL
OAERR_NOTINITIALIZED

Remarks

Since each driver implements different escapes, it is the application's responsibility to ensure that it is sending the escape to the correct driver by comparing the value of the guidFFDriver member of the OADEVICEINSTANCE structure against the value the application is expecting.

The following four methods are TBD.

OAADDevice::Acquire
Obtains control of a device for this application. Due to the nature of how OAAD devices and applications are used Acquire and Unacquire are not believed to be required.

OAADDevice::GetCapabilities
Returns flag data indicating the capabilities of an OAAD device. E.G, the number of axes,.

OAADDevice::Poll
Causes devices to return data and/or to signal events. Used for non-interrupt capable legacy devices. This can be used on any device, and is a no-op for those devices that do not need the prompting. Currently it is not planned to support polling for OAAD devices.

3.3 Data Structures

This section contains information on data structures that are used with OAAD:

3.3.1 OADATAINFO

The OADATAINFO structure carries information describing a device's data format. This structure is used with the OAAD::GetDataFormat method.

```
typedef struct {
   DWORD dwSize;
   DWORD dwReportID;
   DWORD dwReportType;
   LPOADATAFORMAT lpddf;
} OADATAINFO , *LPOADATAINFO ;

typedef const OADATAINFO *LPCOADATAINFO;
```

Members

dwSize
Size of the structure, in bytes. During enumeration, the application may inspect this value to determine how many members of the structure are valid. When the structure is passed to the IOAADDevice::GetObjectInfo method, this member must be initialized to sizeof(OADATAINFO).

wReportID
This unique identifier specifies the ID of the Report.

wReportType
Report type that describes the Report. This value can be one of the following:

OART_FEATURE
The LPOADATAFORMAT lpddf structure is a Feature report that can be read or written over the Control pipe.

OART_INPUT
The LPOADATAFORMAT lpddf structure is a high priority asynchronous Input report that can be read over the Interrupt In pipe.

OART_OUTPUT
The LPOADATAFORMAT lpddf structure is a high priority Output report that can be written over the Interrupt Out or Control pipe.

OART_DI
The LPOADATAFORMAT lpddf structure describes a legacy DirectInput DIDATAFORMAT structure. The wReportType member does not apply and is set to 0.

Remarks

HID devices support three access methods, called *pipes* in HID nomenclature: Input, Output and Feature. Reports are blocks of data that are passed over pipes between the host and the device. HID also allows a device to generate multiple formats for the reports on an individual pipe. Various Reports in a pipe are distinguished using Report ID's. A device uses an Input pipe to send, asynchronously, low latency reports to the host. A host uses an Output pipe to send low latency reports to a device. A Feature pipe is used by a Host to send and receive reports to/from a device.

DirectInput assumes that a device only supports a single report format on an Input pipe. The OADATAINFO structure defines the wReportType member to identify the pipe that a report is associated with and the wReportID member to distinguish a report format on that pipe.

3.3.2 OADEVICEOBJECTINSTANCE

```
typedef struct OADEVICEOBJECTINSTANCEA
{
   DWORD   dwSize;
   GUID    guidType;
   DWORD   dwOfs;
   DWORD   dwType;
   DWORD   dwFlags;
   CHAR    tszName[MAX_PATH];
   DWORD   dwFFMaxForce;
   DWORD   dwFFForceResolution;
   WORD    wCollectionNumber;
   WORD    wDesignatorIndex;
   WORD    wUsagePage;
   WORD    wUsage;
   DWORD   dwDimension;
   WORD    wExponent;
   DWORD    dwReportID;
} OADEVICEOBJECTINSTANCEA, *LPOADEVICEOBJECTINSTANCEA;
```

Members

dwSize
The size of this data structure.

guidType
The GUID associated with this object.

dwOfs
The byte offset in a data structure for a collection of multiple instances of this object.

dwType
Device type that describes this object. It's a combination of OADFT_* flags that describe the object type (button, etc.) with the object instance number in the middle 16 bits. Use the OADFT_GETINSTANCE macro to extract the object instance number. For the OADFT_* flags see the OAADDevice::EnumObjects method.

dwFlags
Flags report other attributes of the data format. The value may be one of the following:
0 in this release.

tszName
Name of the object. For example, "Digital Input"

dwFFMaxForce
Force Feedback Max Force.

dwFFForceResolution
Force Feedback Force Resolution.

wCollectionNumber
The HID link collection to which the object belongs. Or in the parent, in this case we don't have to deal with more than one level deep in the tree. The upper BYTE contains the parent collection number and the lower BYTE contains the collection number for this object. If this object is a member of the topmost collection then the parent number is 0.

wDesignatorIndex
An index which refers to a designator in the HID physical description. This number can be passed to functions in the HID parsing library (Hidpi.h) to obtain additional information about the device object.

wUsagePage
The HID usage page associated with the object, if known. Human Interface Devices will always report a usage page Non-HID devices may optionally report a usage page; if they do not, then the value of this member will be 0.

wUsage
The HID usage associated with the object, if known. Human Interface Devices will always report a usage. Non-HID devices may optionally report a usage; if they do not this field is set to 0.

dwDimension
The dimensional units in which the object's value is reported. if known, or zero if not known. Applications can use this field to distinguish between, for example, the position and velocity of a control.

wExponent
The exponent to associate with the dimension, if known.

dwReportID
Holds the Report Usage ID associated with the data format.

3.3.3 OAPROPHEADER

```
typedef struct OAPROPHEADER
{
   DWORD   dwSize;
   DWORD   dwHeaderSize;
   DWORD   dwObj;
   DWORD   dwHow;
} OAPROPHEADER, *LPOAPROPHEADER;
```

Members

dwSize

Size of the enclosing data structure. This member must be initialized before the structure is used.

dwHeaderSize
Size of the OAPROPHEADER structure.
dwObj
Object for which the property is to be accessed. The value depends on the value specified in the dwHow member.

dwHow
Value that specifies how the dwObj member should be interpreted. This value can be one of the following:

Value	Meaning
OAPH_DEVICE	dwObj member must be zero.
OAPH_BYOFFSET	dwObj member is the offset into the Current data format of the object.
OAPH_BYID	dwObj member is the object type/instance identifier. This identifier is returned in the dwType member of the OADEVICEOBJECTINSTANCE structure returned from a previous call to the OAADDevice::EnumObjects method.

3.3.4 OAPROPDWORD

```
typedef struct OAPROPDWORD
{
   OAPROPHEADER      oaph;
   DWORD             dwData;
} OAPROPDWORD, *LPOAPROPDWORD;
```

Members

oaph
An OAPROPHEADER structure.

dwData
The data value that will be filled in, or set.

3.3.5 OAPROPRANGE

```
typedef struct OAPROPRANGE
{
   OAPROPHEADER      oaph;
   LONG              lMin;
   LONG              lMax;
} OAPROPRANGE, *LPOAPROPRANGE;
```

Members

Oaph
An OAPROPHEADER structure

lMin
Holds the Minimum value for a range.
lMax
Holds the Maximum value for a range.

3.3.6 OADEVICEOBJECTDATA

The OADEVICEOBJECTDATA structure. Contains the device object data.

```
typedef struct OADEVICEOBJECTDATA
{
   DWORD   dwOfs;
   DWORD   dwData;
   DWORD   dwTimeStamp;
   DWORD   dwSequence;
} OADEVICEOBJECTDATA, *LPOADEVICEOBJECTDATA;
```

Members

dwOfs
For GetDeviceData, the offset into the current data format of the object whose data is being reported. The location where the dwData would have been stored if the data had been obtained by a call to the OAADDEVICE::GetDeviceState method. It is an offset relative to the custom data format.

For SendDeviceData, the instance ID of the object to which the data is being sent. This was obtained from the dwType member of an OADEVICEOBJECTINSTANCE structure.

dwData
Data obtained from, or sent to, the device.

dwTimeStamp
The tick count at which the input event was generated, in millisecondes. The current system tick count can be obtained by calling the Win32 GetTickCount function. This value wraps around approximately every 50 days.

For SendDeviceData this member must be 0.

dwSequence
OAAD sequence number for this event. All input events are assigned an increasing sequence number. This allows events from different devices to be sorted in chronological order. Since this value can wrap, care must be taken when comparing two sequence numbers. The OASEQUENCE_COMPARE macro can be used to safely compare two sequences.

For SendDeviceData this member must be 0.

3.3.7 OAEFFESCAPE

```
typedef struct OAEFFESCAPE {
   DWORD      dwSize;
   DWORD      dwCommand;
   LPVOID     lpvInBuffer;
   DWORD      cbInBuffer;
   LPVOID     lpvOutBuffer;
   DWORD      cbOutBuffer;
} OAEFFESCAPE, *LPOAEFFESCAPE;
```

Members

dwSize
Size of this structure in bytes. Must be initialized before the structure is used.

dwCommand
Driver-specific command number. Consult the driver documentation for a list of valid commands.

lpvInBuffer
Buffer containing the data required by the operation.

cbInBuffer
The number of bytes in the lpvInBuffer buffer.

lpvOutBuffer
Buffer that receives the operation's output data.

cbOutBuffer
On entry, the number of bytes in the lpvOutBuffer buffer. On exit, the number of bytes actually produced by the command.

Note: This data structure is not currently used by OAAD.

3.3.8 OADEVICEINSTANCE

```
typedef struct OADEVICEINSTANCEA
{
   DWORD      dwSize;
   GUID       guidInstance;
   GUID       guidProduct;
   DWORD      dwDevType;
   CHAR       tszInstanceName[MAX_PATH];
   CHAR       tszProductName[MAX_PATH];
   GUID       guidFFDriver;
   WORD       wUsagePage;
   WORD       wUsage;
} OADEVICEINSTANCE, *LPOADEVICEINSTANCE;
```

Members

dwSize
The size of this data structure, in bytes. Must be initialized before the structure is used.

guidInstance
A unique identifier for the instance of the device. An application may save the instance GUID in a configuration file, or the registry, for use at a later time. Instance GUIDs are specific to a particular computer. An instance GUID obtained from one computer is not related to instance GUIDs on another computer.

guidProduct
A unique identifier for the product. This identifier is established by the manufacturer of the device, or the creator of an OAAD Device Object.

dwDevType
The type of the device. The least-significant byte of the device type description code specifies the device type. The next-significant byte specifies the device subtype. The value can be one of the following types combined with their respective subtypes, and optionally with OADEVTYPE_HID, for Human Interface Devices.

OADEVTYPE_COINDOOR

OADEVTYPE_BILLCHANGER

OADEVTYPE_GCICARD

OADEVTYPE_HID
The device uses the Human Interface Device (HID) protocol.

tszInstanceName
The friendly name for the instance. For example, “Joystick 1.”

tszProductName
The friendly name for the product.

guidFFDriver
The unique identifier for the driver being used for force feedback. This identifier is established by the manufacturer of the driver.

wUsagePage
If the device is a HID device, this member contains the HID usage page code.

wUsage
If the device is a Human Interface Device, this member contains the HID Usage Report code.

3.3.9 HID Report Data Structures

We will define Generic Analog Input, Generic Digital Input, Generic Digital Output. These are supported in the collections for the GCI card. These structures are provided for the convenience of the developer. The information needed to map the report data can also be obtained from Enumerate Objects and Enumerate Data Formats.

These are the basic structures associated with the Usage reports. Note that in cases of multiple instances the additional instance data will follow the first instance data. E.g. GCI_COIN_DOOR_DROPCOUNT with two instances would look like this:

```
BYTE bytReportID ;
BYTE nDropCount1 ;
BYTE nDropCount2 ;
        or
BYTE bytReportID ;
BYTE nDropCount[2] ;

typedef struct _tagCoinNumDoors
{
        BYTE    bytReportID ;
        BYTE    nCoinDoors ;
} GCI_NUM_COIN_DOORS ;

typedef struct _tagCoinDoorDropCount
{
        BYTE    bytReportID ;
        BYTE    nDropCount ;
} GCI_COIN_DOOR_DROPCOUNT ;

typedef struct _tagCoinDoorStart
{
        BYTE    bytReportID ;
        BYTE    nCoinStart ;
} GCI_COIN_DOORSTART ;

typedef struct _tagCoinDoorSrvc
{
        BYTE    bytReportID ;
        BYTE    nCoinService ;
} GCI_COIN_DOORSRVC ;

typedef struct _tagCoinTilt
{
        BYTE    bytReportID ;
        BYTE    nCoinTilt ;
} GCI_COIN_TILT ;

typedef struct _tagCoinTest
{
        BYTE    bytReportID ;
        BYTE    nCoinTest ;
} GCI_COIN_TEST ;

typedef struct _tagCoinDoorHIDControlFeatureReport
{
        BYTE    bytReportID ;
        BYTE    bytControl ; // Coin lockout control byte
} GCI_COIN_DOOR_LOCK_FEAT ;
```

3.3.10

OADEVCAPS

```
typedef struct OADEVCAPS
{
   DWORD   dwSize;
   DWORD   dwFlags;
   DWORD   dwDevType;
   DWORD   dwAxes;
   DWORD   dwButtons;
   DWORD   dwPOVs;
   DWORD   dwFFSamplePeriod;
   DWORD   dwFFMinTimeResolution;
   DWORD   dwFirmwareRevision;
   DWORD   dwHardwareRevision;
   DWORD   dwFFDriverVersion;
} OADEVCAPS, *LPOADEVCAPS;
```

Members

dwSize
The size of this structure in bytes. This member must be initialized by the application before calling the OAADDevice::GetCapabilities method.

dwFlags
Flags associated with the device. It can be a combination of the following:

OADC_ATTACHED
The device is physicall attached.

OADC_EMULATED
The device functionality is emulated.

OADC_POLLEDDATAFORMAT
At least one object in the current data format is polled rather than interrupt-driven. For these objects, the application must explicitly call the OAADDevice2::Poll method to obtain data. Note: this is NOT a requirement for the initial implementation of OAAD.

OADC_POLLEDDEVICE
At least one object on the device is polled rather than interrupt-driven. For these objects, the application must explicitly call the OAADDevice2::Poll method to obtain data. HID devices may contain a mixture of polled and non-polled objects. Note: this is NOT a requirement for the initial implementation of OAAD.

dwDevType
The device type specifier. This member can contain values identical to those in the dwDevType member of the OADEVICEINSTANCE structure.

dwAxes
Number of axes available on the device.

dwButtons
Number of buttons available on the device.

dwPOVs

Number of point-of-view controllers available on the device.

dwFFSamplePeriod
The minimum time between playback of consecutive raw force commands.

dwFFMinTimeResolution
The minimum amount of time, in microseconds, that the device can resolve. The device rounds any times to the nearest supported increment. If the value of dwFFMinTimeResolution is 1000 then the device would round any times to the nearest millisecond.

dwFirmwareRevision
The firmware revision of the device.
dwHardwareRevision
The hardware revision of the device.

dwFFDriverVersion
The version number of any device driver.

Remarks

The semantics of version numbers are left to the manufacturer of the device. They are only guaranteed to have larger numbers for newer versions. The suggested format is 0x0000HHLL where HH is the major (high order) version and LL is the minor (low order) version.

3.3.11 OAOBJECTDATAFORMAT

The OAOBJECTDATAFORMAT structure. Used to return data format information.

```
typedef struct _OAOBJECTDATAFORMAT
{
   const GUID *pguid;
   DWORD   dwOfs;
   DWORD   dwType;
   DWORD   dwFlags;
} OAOBJECTDATAFORMAT, *LPOAOBJECTDATAFORMAT;
```

Members

pguid
Unique identifier for the input source. When requesting a data format, making this member NULL indicates that any type of object is permissible.

dwOfs
Offset within the data packet where the data for the input source will be stored. This value must be a multiple of four for DWORD size data, such as axes. It can be byte-aligned for buttons.

dwType
Device type that describes the object. It is a combination of the following flags describing the object type (axis, button, and so forth) and containing the object-instance number in the middle 16 bits. When requesting a data format, the instance portion must be set to OADFT_ANYINSTANCE to indicate that any instance is permissible, or to OADFT_MAKEINSTANCE(n) to restrict the request to instance n. See the examples under Remarks.

OADFT_ANALOG_INP
The object must be an OAAD Generic Analog Input

OADFT_DIGITAL_INP
The object must be an OAAD Generic Digital Input

OADFT_OPTICAL_INP
The object must be an OAAD Generic Optical Input

OADFT_COINDOOR
The object must be an OAAD CoinDoor Input

OADFT_DIGITAL_OUT
The object must be an OAAD Generic Digital Output

OADFT_COINDOOR_OUT
The object must be an OAAD CoinDoor Output

OADFT_PSHBUTTON
The object must be a push button

OADFT_BUTTON
The object must be a button

dwFlags
Must be 0 for this release.

3.3.12 OADATAFORMAT

The OADATAFORMAT structure. Used to return data format information.

```
typedef struct _OADATAFORMAT
{
   DWORD   dwSize;
   DWORD   dwObjSize;
   DWORD   dwFlags;
   DWORD   dwDataSize;
   DWORD   dwNumObjs;
   LPOAOBJECTDATAFORMAT rgodf;
} OADATAFORMAT, *LPOADATAFORMAT;
typedef const OADATAFORMAT *LPCOADATAFORMAT;
```

Members

dwSize
Size of this structure, in bytes. Must be initialized before the structure is used.

dwObjSize
Size of the OAOBJECTDATAFORMAT structure, in bytes.

dwFlags
Flags that describe other attributes of the data format. This value can be one of the following:

OADFT_ANALOG_INP
The object must be an OAAD Generic Analog Input

OADFT_DIGITAL_INP
The object must be an OAAD Generic Digital Input

OADFT_OPTICAL_INP
The object must be an OAAD Generic Optical Input

OADFT_COINDOOR
The object must be an OAAD CoinDoor Input

OADFT_DIGITAL_OUT
The object must be an OAAD Generic Digital Output

OADFT_COINDOOR_OUT
The object must be an OAAD CoinDoor Output

OADFT_PSHBUTTON
The object must be a push button

OADFT_BUTTON
The object must be a button

dwDataSize
Size of a data packet returned by the device, in bytes. Must be a multiple of 4 and must exceed the largest offset value for an object's data within the data packet.

dwNumObjs
Number of objects in the rgodf array.

rgodf
Address of an array of OAOBJECTDATAFORMAT structures. Each structure describes how one object's data should be reported in the device data.

Unresolved Issues

Notes:

Additional error codes, definitions, remarks and examples will be added in future revisions of this document.

---

4 Appendix A Narrative Sample Application Description

4.1 Introduction

This document illustrates how the OAAD SDK extensions to Microsoft's DirectInput are to be used.

The OAAD SDK is intended to supplement Microsoft's DirectInput by providing support for devices that DirectInput either does not currently support, or only partially supports. The OAAD SDK is intended to work along side DirectInput. Those devices that are fully supported by DirectInput should continue to use it. This document makes the assumption that the reader is familiar with Microsoft's COM and the DirectX API, specifically the DirectInput portion.

4.2 OAAD Devices

This section contains information showing a simple Credit Acceptor connected to a GCI card. Specifically a Bill Acceptor example for sending a Report to the device using the OAAD SDK extensions:

4.2.1 Device Setup

Your application must obtain a COM interface for each device with which it expects to interact. Each device must also be prepared before use. This requires, at a minimum, obtaining the data format and acquiring the device. In addition, you may also wish to carry out other configuration tasks, e.g. getting information about the devices and changing their properties.

The following tasks are part of the setup process. Some of these steps are always required. Other steps may only be needed if you require additional information about the device or need to change default values.

- Create the OAAD device instance (required). See Creating an OAAD Device.
- Get the device capabilities (optional).
- Enumerate the keys, buttons, reports, etc. on the device (optional). See Device Object Enumeration.
- Set the cooperative level (recommended).
- Enumerate the data format(s) (required).
- Set the device properties (optional).
- When ready to read or write data, acquire the device. See Acquiring Devices.

4.2.2 Creating an OAAD instance

The first thing that an OAAD arcade application must do is to create an instance of an OAAD object Interface. Since many arcade applications are written in 'C' we will show the 'C' approach to calling the OAAD methods.

```
HRESULT hResult = OAADCreate( hInstance,
                                        dwVersion,
                                        &m_gpOAAD,
                                        punkOuter );
if( FAILED(hResult) )
{
        /* Failed to create the OAAD instance notify the user in an appropriate manner, this would be a fatal error */
        printf( "Failed to create OAAD instance, 0x%08.8x\nPress Enter to Exit\n", hResult ) ;
        getchar() ;
```

```
                exit(1) ;
        }
```

4.2.3 Enumerating OAAD Devices

Once we have an instance of an OAAD interface there are several things that an arcade application will want to do. One of the things would be to call the OAAD::GetVersion method on that interface and verify that it is the minimum version that the application requires to function.

Next we need to enumerate the OAAD devices that may be present on the system. These devices may include GCI cards, trackballs, buttons, coin doors, etc. Note that GCI cards may have several devices attached to them.

In order for an application to enumerate the devices that are attached we will need to supply a Windows callback function. This function will be called once for each device that is found on the system.

Here is an example of such a callback function:

```
BOOL CALLBACK OAEnumDevicesProc(
 LPCOADEVICEINSTANCE lpdoa,
 LPVOID pvRef )
{
        DWORD dwDevNum ;
        CHAR *pstrTmp  = NULL ;
        CHAR *pstrTmp1 = NULL ;
        CHAR * pstrInstanceName = NULL ;
        CHAR szBuffer[1024] ;
        dwDevNum = *(DWORD *)pvRef ;

        /* Here we would take the enumerated information and build whatever internal tables of GUIDs, etc., are
         important to the application.  The GUID associated with the type of device would be used as needed by
         the application. */
        if( OADEVTYPE_BILLCHANGER == (~OADEVTYPE_HID & lpdoa->dwDevType) )
        {
                pstrTmp1 = "BillChanger" ;
        }
        if( OADEVTYPE_COINDOOR == (~OADEVTYPE_HID & lpdoa->dwDevType) )
        {
                pstrTmp1 = "CoinDoor" ;
        }

        pstrInstanceName = (CHAR *)lpdoa->tszInstanceName ;
        /* Save the GUID for the desired device (for now) */
        if( NULL != strstr( pstrInstanceName, "GCI2" ) )
        {
                /* Found the GCI2 sub-string in this entry */
                gGUIDProduct = lpdoa->guidProduct ;
                gbGUIDSet            = TRUE ;
```

```
        }
        /* Build a buffer for  output, this will just display the information in the DOS box. */
        sprintf( &szBuffer[0], "Device %d Type: '%s'\nInstanceName: '%s'\nProductName: '%s'\n\n", dwDevNum, pstrTmp1,
                                                        lpdoa->tszInstanceName, lpdoa->tszProductName ) ;
         printf( "%s", szBuffer ) ; /* Output the string */
        return TRUE ; /* Keep on enumerating, if you want to stop before all devices are really enumerated return FALSE  */
}
```

Now we need some code that will set up and make the request for the enumeration.

m_gpOAAD is a global, in this example, that holds the pointer to the OAAD Interface object that was created earlier.

```
        DWORD dwDevType = 0 ;                   /* All device types */
        DWORD dwRef     = 2345 ;                /* Purely arbitrary, use something that would be useful when processing in the
callback function */
        DWORD dwFlags   = 0 ;
        HRESULT hr ;
        printf( "Enumerating Devices\n" ) ;
        hr = IOAAD_EnumDevices(m_gpOAAD, dwDevType, OAEnumDevicesProc, &dwRef, dwFlags) ;
        if( FAILED(hr) )
        {
                printf( "EnumDevices failed with error 0x%08.8x", hr ) ;
                return hr ;
        }
```

4.2.4 Creating OAAD Device Instances

Once we have successfully enumerated the OAAD devices we should create instances of the devices that are desired. In this example we have setup, in the callback function, the GUID of the GCI2 sample device object. We will now create an instance of this OAAD Device Interface. The gGUIDProduct global was set in the enumerate devices callback function above.

The **OAAD::CreateDevice** method is used to obtain a pointer to the **IOAADDevice** interface. Methods of this interface are then used to manipulate the device and to send and receive data.

The following example, where *lpdi* is a pointer to the **IOAAD** interface, creates a Bill Acceptor device:

```
LPOAADDEVICE  lpoaadBillAcceptor;
lpdi->CreateDevice(GUID_BillAcceptor, &lpoaadBillAcceptor, NULL);
```

The first parameter in **IOAAD::CreateDevice** is an instance GUID that identifies the instance of the device for which the interface is to be created. NOTE: OAAD currently does not have any predefined GUIDs.

Use the instance GUID for the device returned by **IOAAD::EnumDevices**. The instance GUID for a device will always be the same for a given installation on a machine. Note that you can present a list of the enumerated devices and allow the user to select a device from a list of those enumerated, if that is appropriate for your application. You can then save the GUID associated with the selected device to a configuration file and use it in future sessions, without needing to perform the enumeration step. This would be useful for configuring a system where certain devices will not be used in certain configurations.

```
        if( NULL == m_gpOAADDev )
```

```
{
        /* Create the device instance here */
        GUID rrguid = gGUIDProduct ;
        LPOAADDEVICE pOADevice ;
        LPUNKNOWN pUnkwn = NULL ;
        /* p - 'this' pointer (pointer to OAAD interface instance,
        a – RefGUID(the ID of the target device)  b - LPOAADDEVICE
        c - LPUNKNOWN*/
        HRESULT hr = IOAAD_CreateDevice(m_gpOAAD, &rrguid, &pOADevice, pUnkwn) ;
        if (FAILED(hr ) )
        {
                printf("Error %x creating OAADDevice Object!", hr);
                return hr ;
        }
        /* For this example we will save the pointer to the device interface in a global */
        m_gpOAADDev = pOADevice ;
}
```

4.2.5 Enumerating Device Data Formats

Now that we have created an instance of the device we want to do something interesting. In order for an OAAD application to communicate with a device it needs to know the data format that will be used. In DirectInput the application sets the data format. In OAAD the device specifies the data format. The application must make a call to the **OAADDevice::EnumDataFormats** method to determine the data formats that are supported.

The EnumDataFormats method functions in a manner similar to OAAD::EnumDevices, and requires a Windows callback function that will be called once for each data format that the device supports.

Here is one such callback function:

```
BOOL CALLBACK OAEnumDataFmts(
 LPCOADATAINFO lpdoa,
 LPVOID pvRef )
{
        DWORD dwFmtNum = 0 ;
        DWORD dwReportID = 0 ;
        DWORD dwReportType      = 0 ;
        DWORD dwDataSize        = 0 ;
        DWORD dwNumObjs         = 0 ;
        LPOAOBJECTDATAFORMAT pTmp = NULL ;
        char * pstrReportID ;
        char * pstrReportType ;
        char szOffsets[16384] ;
        DWORD dwIdx = 0 ;
        if( NULL == lpdoa )
```

```
{
        printf( "NULL pointer passed to EnumDataFmts\n" ) ;
        return FALSE ;
}

/* Attempt to get the reference number that was passed in, if you need to use it for anything. */
if( pvRef )
        dwFmtNum = *(DWORD *)pvRef ;
else
        printf( "pvRef is NULL in enum data formats callback, using 0.\n" ) ;

/* Take apart the structures and report what we have.. */
dwReportID      = lpdoa->dwReportID ;
dwReportType    = lpdoa->dwReportType ;
dwDataSize      = lpdoa->lpddf->dwDataSize ;
dwNumObjs       = lpdoa->lpddf->dwNumObjs ;
pTmp = lpdoa->lpddf->rgodf ;

/* For this example we will just output to the 'DOS' window */
pstrReportID = "Unknown" ;
pstrReportType = "Unknown" ;
switch( dwReportID )
{
        case BILL_ACCEPT_CONTROL_OUTPUT_REPORT:
                pstrReportID    = "BILL_ACCEPT_CONTROL_OUTPUT_REPORT" ;
                break ;

        case BILL_ACCEPT_CONTROL_FEATURE_REPORT:
                pstrReportID    = "BILL_ACCEPT_CONTROL_FEATURE_REPORT" ;
                break ;

        case BILL_ACCEPTED_COUNT_INPUT_REPORT:
                pstrReportID    = "BILL_ACCEPTED_COUNT_INPUT_REPORT" ;
                break ;
};

switch( dwReportType )
{
        case OART_OUTPUT:
                pstrReportType = "OART_OUTPUT" ;
                break ;

        case OART_FEATURE:
```

```
                        pstrReportType = "OART_FEATURE" ;
                        break ;

                case OART_INPUT:
                        pstrReportType = "OART_INPUT" ;
                        break ;
        };

        /* Build a string with the offsets for the data here...*/
        sprintf( szOffsets, "Ofsts: " ) ;
        for( dwIdx = 0 ; dwIdx < dwNumObjs ; dwIdx++ )
        {
                char szTmp[256] ;
                sprintf( szTmp, "(%02d) %d ", dwIdx, pTmp->dwOfs ) ;
                ++pTmp ;
                strcat( szOffsets, szTmp ) ;
        }
        printf( "Dev Data Format %d\nReptID %s\nReptType %s\nDataSz %d Bytes\nNumObjs %d\n%s\n\n", dwFmtNum,
                pstrReportID, pstrReportType, dwDataSize, dwNumObjs, szOffsets ) ;

        return TRUE ; /* TRUE to keep enumerating, FALSE to stop */
}
```

Now that we have our Data Formats Enumeration callback function we need to set up and call the EnumDataFormats method.

```
        DWORD dwReportType      = 0 ; /* All report types */
        DWORD dwReportID = 0 ; /* All report ids */

        DWORD dwRef             = 1234 ; /*Arbitrary for the example*/
        DWORD dwFlags           = 0 ; /* No modifier flags */
        HRESULT hr ;
        if( NULL == m_gpOAADDev )
        {
                printf( "Must Create Device first\n" ) ;
                return ;
        }

        printf( "Enumerating Data Formats\n" ) ;

        hr = IOAADDevice_EnumDataFormats(m_gpOAADDev, dwReportType, dwReportID,
                                                OAEnumDataFmts, &dwRef, dwFlags) ;
        if( FAILED(hr) )
        {
```

```
            printf( "EnumDataFormats Failed with error code 0x%08.8x\n", hr ) ;
    }
```

At this point we would want to do something useful with our data format information, such as controlling the device, or getting data from it. That will be illustrated later. For now we will show how you cleanup when your application is finished using OAAD.

When each instance of an OAAD or OAADDevice object was created the arcade application automatically acquired a reference to the object. Unlike traditional pointers you do not want to simply delete this object. You must call the Release method on each valid interface pointer that you have created in your application.

For completeness it is an excellent idea to immediately set the pointer to NULL after calling the Release method. Of course in this simple example the setting of the pointer to NULL could be eliminated, but it is a good habit to get into, as it makes finding elusive bugs easier.

4.2.6 Cleaning Up and Shutting Down

```
    /* Finally we need to release the object(s) */
    if( m_gpOAADDev )
            IOAADDevice_Release(m_gpOAADDev) ;
    m_gpOAADDev = NULL ;


    if( m_gpOAAD )
            IOAAD_Release(m_gpOAAD) ;
    m_gpOAAD = NULL ;


    printf( "C program calling OAAD SDK finished\n" ) ;
    return 0 ;
```

4.2.7 Getting Device Capabilities

Before you begin asking for input from a device, you may need to find out something about its capabilities. Does the joystick have a point-of-view hat? Is the mouse currently attached to the user's machine? Such questions are answered with a call to the **IOAADDevice::GetCapabilities** method, which returns the data in an **OADEVCAPS** structure. As with other such structures in OAAD, you must initialize the **dwSize** member before passing this structure to the function.

Here's an example that checks capabilities of a device:

```
//  LPOADEVICE  lpoaadDevice;  // initialized previously

OADEVCAPS  OADeviceCaps;
HRESULT    hr;

OADeviceCaps.dwSize = sizeof(OADEVCAPS);
hr = lpoaadDevice->GetCapabilities(&OADeviceCaps);
// Interpret the dwFlags member in the context of the device, as appropriate
```

Another way to check for a certain feature is to call **IOAADDevice::GetObjectInfo** for that object. If the call returns OAERR_OBJECTNOTFOUND, the object is not present.

4.2.8 Cooperative Levels

The cooperative level of a device determines how the input is shared with other applications and with the Windows system. You set it by using the **IOAADDevice::SetCooperativeLevel** method, as in this example:

```
lpdiDevice->SetCooperativeLevel(hwnd, OASCL_NONEXCLUSIVE | OASCL_FOREGROUND)
```

Note

This OAAD functionality operates in the same manner as Microsoft's DirectInput. Refer to the DirectInput documentation for more details.

4.2.9 Device Data Formats

Since the devices that the OAAD SDK supports typically supply data in multiple "reports" your application will need to enumerate the device data formats in much the same way you enumerate devices.

The **IOAADDevice::EnumDataFormats** method enumerates the data formats that are associated with a device.

Like **IOAAD::EnumDevices**, the **EnumDataFormats** method uses a callback function that gives you the chance to do other processing on each data format. You can, for example, add it to a list or create a corresponding element on a user interface.

Here's a callback function that simply extracts the report ID of each data format so that it can be added to a list or array. This standard callback is documented under the placeholder name **EnumDataFormatsProc**, but you can give it any name you like. Remember, this function is called once for each object enumerated.

```
char  szName[MAX_PATH];

BOOL CALLBACK OAADEnumDataFormatsProc(
                      LPCOADATAINFO lpoaadDI,
                      LPVOID pvRef)
{

  DWORD dwReportId ;
  dwReportId = lpoaadDI->dwReportID ;
  // Now add dwReportId to a list or array
  .
  .
  .
  return OAENUM_CONTINUE;
}
```

The first parameter points to a structure containing information about the object. This structure is created for you by OAAD.

The second parameter is an application-defined pointer to data, equivalent to the second parameter to **EnumDataFormats**. In this example, the parameter is not used.

The return value in this case indicates that enumeration is to continue as long as there are still objects to be enumerated.

Here's how to use the **EnumDataFormats** method.

```
lpoaadDevice->EnumObjects(dwReportType, dwReportID, OAADEnumDataFormatsProc, NULL,
OADFT_NONE);
```

The first parameter is the report type filter. If it is zero all report types will be enumerated. Otherwise, it is an OART_* value (see OADATAINFO in the OAAD SDK documentation), indicating the report type that should be enumerated.

The second parameter is the report ID filter. If this is zero, all report Ids are enumerated. Otherwise, it is a value that indicates the report ID that should be enumerated.

The third parameter is the address of the callback function.

4.2.10 Device Properties

Refer to the Microsoft DirectInput documentation for a general discussion of device properties.

4.2.11 Acquiring Devices

Acquiring a OAAD device means giving your application access to it. As long as a device is acquired, OAAD will make its data available to your application. If the device is not acquired, you may manipulate its characteristics but not obtain any data.

Refer to the DirectInput documentation for more details of how acquiring devices functions. Note that many OAAD devices do not need to be 'acquired' as they are not shared system devices.

4.2.12 Enumerating Objects

Once a device instance has been created the application should enumerate the Objects that are on the device. These include things like Generic Digital Inputs and Outputs, Generic Analog Inputs, Coin Door Inputs and Outputs, etc. The application can build a set of data buffers that will hold information for any of the objects that the application needs to interact with. Refer to the 'C' or 'C++' example applications for details of one such implementation.

4.2.13 Enumerating Data Formats

In addition to enumerating the Objects on a device the application should also enumerate the data formats that are supported. By using the information returned for the Data Format associated with a given object the application can assure that the proper amount of memory is reserved to hold the data that is being sent or received. In addition the layout of the data within the buffer is described by the Data Format. Refer to the 'C' or 'C++' example applications for details of one such implementation.

4.3 Device Data

This section covers the basic concepts of getting data from OAAD devices.

- Buffered and Immediate Data
- Time Stamps and Sequence Numbers
- Polling and Events
- Relative and Absolute Axis Coordinates

4.3.1 Buffered and Immediate Data

OAAD supports two types of data access: buffered and immediate. Buffered data is a record of events that is stored until an application retrieves it. Immediate data is a snapshot of the current state of a device at a given point in time.

The OAAD data access methods operate in a manner similar to the DirectInput methods of the same name, with the extensions to support multiple reports from the OAAD device. Refer to the Microsoft DirectInput documentation for a general discussion of buffered and immediate data and the OAAD SDK documentation for details of the OAAD extensions. **Note: Buffered data is only supported if the associated OAAD Device Object implements it.**

4.3.2 Output Data

Human Interface Devices may accept output as well as generating input. The **OAAD::SetDeviceData** method is used to send packets of low latency, buffered data to such devices. The **OAAD::SetDeviceState** method is used to send instantaneous data to the device. In either case the specific report type and ID are indicated by the OADATAINFO structure that is passed by the application.

SetDeviceData may be viewed as **OAAD::GetDeviceData** in reverse. Like that method, it uses the **OADEVICEOBJECTDATA** structure as the basic unit of data. In this case, however, the **dwOfs** member contains the instance ID of the device object associated with the data, rather than its offset in the data format for the device. (Because offset identifiers exist only for device objects that provide input in the selected data format, an object that only accepts output may not even have an offset.) The **dwData** member contains whatever data is appropriate for the object. The **dwTimeStamp** and **dwSequence** members are not used and must be set to zero.

To send data to the device, you first set up an array of **OADEVICEOBJECTDATA** structures, fill the required number of elements with data, then pass its address and the number of elements used to **SendDeviceData**. Data for different device objects is combined into a single packet that is then sent to the device.

The form of the data packet is specific to the device, as is the treatment of unused fields in the packet. The data and treatment of the data is based on the HID Usage Table for that device. Some devices may treat fields as optional, meaning that if no data is supplied, the state of the object remains unchanged. Usually, all of the fields are significant, even when you do not specifically supply data for them. Any data that is not supplied will be assumed to take on the default value, e.g. 0. You can override this behavior by using the OASDD_CONTINUE flag, which will cause data for other items to be the value that was most recently sent for those items.

4.3.3 Example of Controlling an OAAD Device

#define BILL_ACCEPT_CONTROL_OUTPUT_REPORT 0x10

```
void SetDenominations(void)
{
DWORD                cdod = 3;                  // number of items
OADATAINFO           OaDI;
HRESULT              hRes;

// Clear data members
ZeroMemory(oaDI, sizeof(OaDI));

OaDI.dwReportID   = BILL_ACCEPT_CONTROL_OUTPUT_REPORT ; //The identifier of the report
OaDI.dwReportType = OART_OUTPUT ; // Which HID pipe is to be used to pass the Report
data to the device.
// Here we need to specify the data structure for the report..
typedef struct _tagHID_BILL_ACCEPT_CONTROL_OUTPUT_REPORT {
    BYTE  bytReportID;          // HID Report ID goes here
    BYTE  byt1;                 // Second byte of the report
    BYTE  byt2;                 // Third byte of the report
```

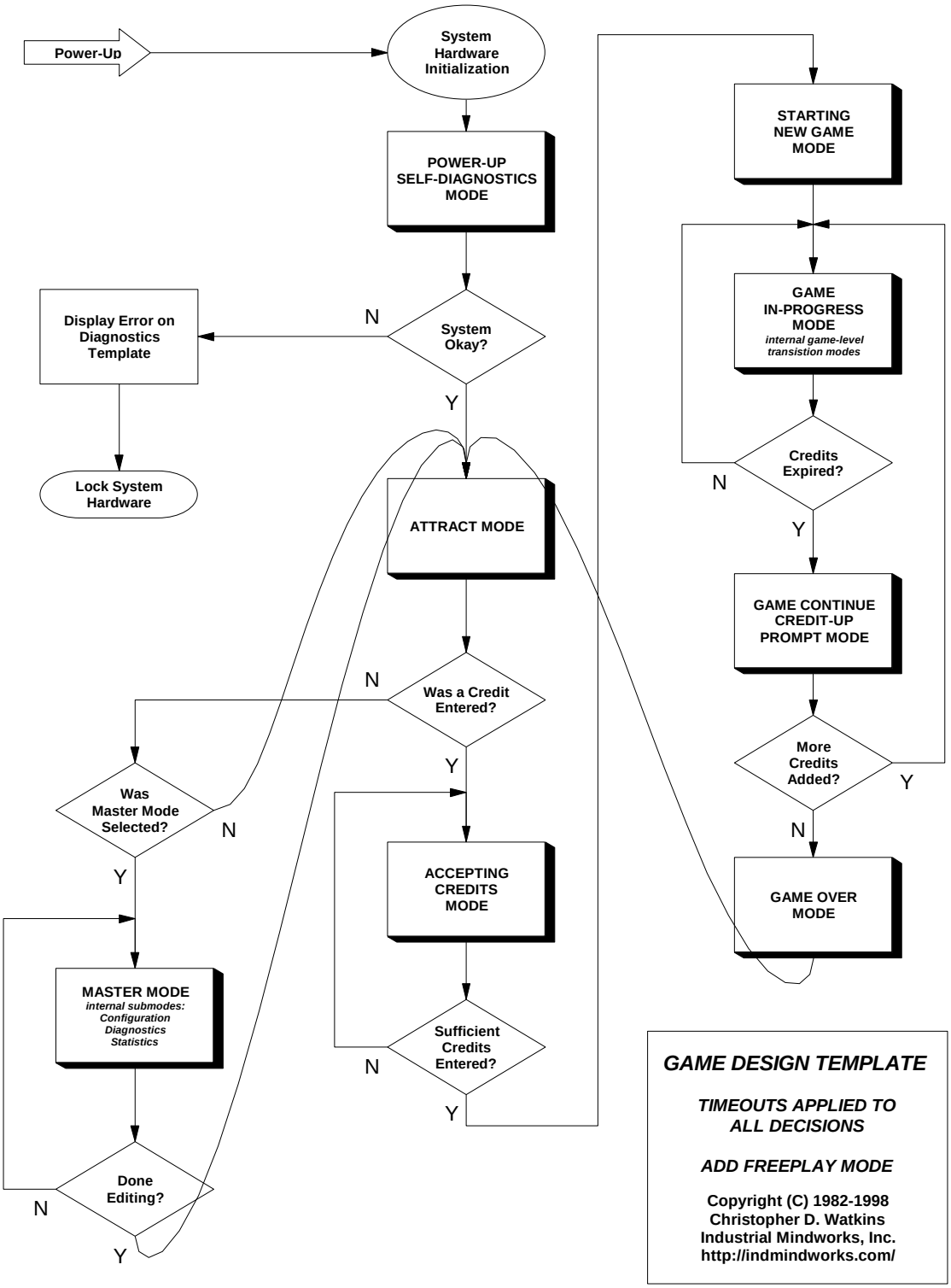
```
    BYTE  bytPad;               // Must be DWORD multiple in size
} HID_BILL_ACCEPT_CONTROL_OUTPUT_REPORT;

HID_BILL_ACCEPT_CONTROL_OUTPUT_REPORT theHIDData ;
theHIDData.bytReportID = BILL_ACCEPT_CONTROL_OUTPUT_REPORT ;
theHIDData.byt1 = 0xe1 ; // Set bits for Bill Denom 0 – 2 and Device Enable
theHIDData.byt2 = 0x00 ; // Clear all Coupon Denom bits.

OAOBJECTDATAFORMAT rgodf[ ] =
{
  { &GUID_ReportID, FIELD_OFFSET(HID_BILL_ACCEPT_CONTROL_OUTPUT_REPORT, bytReportID),
      OADFT_ANYINSTANCE, 0, },
  { &GUID_BACOR1, FIELD_OFFSET(HID_BILL_ACCEPT_CONTROL_OUTPUT_REPORT, byt1),
      OADFT_ANYINSTANCE, 0, },
  { &GUID_BACOR2, FIELD_OFFSET(HID_BILL_ACCEPT_CONTROL_OUTPUT_REPORT, byt2),
      OADFT_ANYINSTANCE, 0, }
};
#define numObjects (sizeof(rgodf) / sizeof(rgodf[0]))

      OADATAFORMAT theDIDDataFormat =
      {
          sizeof(OADATAFORMAT),       // This structure
          sizeof(OAOBJECTDATAFORMAT), // Size of object data format
          DIDF_ABSAXIS,               //
          sizeof(HID_BILL_ACCEPT_CONTROL_OUTPUT_REPORT),  // Device data size
          numObjects,                 // Number of objects
          rgodf,                      // Here are the actual objects
      };

      OaDI.lpddf = &theDIDDataFormat ;

      hRes = IOAADDevice_SetDeviceState( cbData, &theHIDData, &OaDI);
}
```

Industrial Mindworks®, Inc.

PC-based Computer Graphics and Simulation Software & Hardware Development for Training, Education, Medicine and Entertainment

PARTIAL "GENERIC" SIMULATION GAME DESIGN TEMPLATE

http://www.IndMindworks.com/


Power-Up
System Hardware Initialization
POWER-UP SELF-DIAGNOSTICS MODE
System Okay?
N
Y
Display Error on Diagnostics Template
Lock System Hardware
ATTRACT MODE
Was a Credit Entered?
N
Y
Was Master Mode Selected?
N
Y
MASTER MODE
internal submodes:
Configuration
Diagnostics
Statistics
Done Editing?
N
Y
ACCEPTING CREDITS MODE
Sufficient Credits Entered?
N
Y
STARTING NEW GAME MODE
GAME IN-PROGRESS MODE
internal game-level transistion modes
Credits Expired?
N
Y
GAME CONTINUE CREDIT-UP PROMPT MODE
More Credits Added?
Y
N
GAME OVER MODE
GAME DESIGN TEMPLATE
TIMEOUTS APPLIED TO ALL DECISIONS
ADD FREEPLAY MODE
Copyright (C) 1982-1998
Christopher D. Watkins
Industrial Mindworks, Inc.
http://indmindworks.com/

SYSTEM SOFTWARE LAYERS OF ABSTRACTION

The Application:	The Game
High-level libraries:	Game API for Game Developers
Mid-level libraries:	Control Inputs API (3dof Boom, Triggers, Buttons), Special Inputs API (Master Mode button, Coin-Ops), Graphics Outputs API (2D/3D Rendering, Fonts, Special Effects, FLC Player) Audio Outputs API (WAV Sound Effects, MIDI Music), Special Outputs API (Motion-platform Control System) Dynamics Simulation/Collisions API, Artificial Intelligence API
Low-level libraries:	Device Drivers (Special I/O-card Driver, OEM-supplied APIs and drivers for Graphics and Sound)

SYSTEM SOFTWARE DISPLAY TYPES

System Software Display Types represent the component types of “documents” available for processing by the graphics system.

Screen Type: Either a static 2D BMP Image or an active 2D FLC Animation.

Overlay Type: Either a 2D BMP Image Overlay or a 3D Rendering Overlay.

Overlays may be Static or Active.
Static overlays are one-time renderings to the screen.
Active overlays are Templates whose fields are updated by subprocesses (such as the modification of a machine configuration parameter with the boom controller).

Real-Time Type: Game in operation writing to a window with 3D Rendering.

SYSTEM OPERATIONAL MODES

System Operational Modes are accompanied by an ordered series of Screen Types and Overlay Types (as well as Sounds and Music).

Screen	Overlay	Real-Time
Power-Up Self-Diagnostics Mode		
Diagnostics Bkgnd Image	Diagnostics Template Image	
Attract Mode		
Title Bkgnd Image	Insert Credits Image	
Acknowledgments Image	Insert Credits Image	
Game Play #1 Animation	Insert Credits Image	
High Score Bkgnd Image	Insert Credits Image + High Score Data Template	
High Score Multicredit Bkgnd Image	Insert Credits Image + High Score Multicredit Data Template	
Game Play #2 Animation	Insert Credits Image	
Accepting Credits Mode		
Credit-up Bkgnd Image	Credit Count Template Image	
Starting New Game Mode		
Instruction Animation		
Game Entry Animation		
Game In Progress Mode		
Visor Bkgnd Image	Visor Image Template	Game Active
Game Level Transition Mode		
Transition Animation	Life/Score Template	
Game Continue Credit-up Prompt Mode		
Continue Play Bkgnd Image	Continue Play Countdown/ Credit Count Template	
Game Over Mode		

Game Over Animation

Insert Initials High Score Bkgnd Image	Insert Initials High Score Entry Template
High Score Bkgnd Image	Insert Credits Image + High Score Data Template
High Score Multicredit Bkgnd Image	Insert Credits Image + High Score Multicredit Data Template

Master Mode

Machine Configuration Bkgnd	Machine Configuration Template
Machine Self-Diagnostics Bkgnd	Machine Self-Diagnostics Template
Machine Statistics Bkgnd	Machine Statistics Template

5 Appendix B: Definitions and Error Codes

5.1 General Definitions

OAENUM_STOP Returned from enumeration callback function to stop enumerating.
OAENUM_CONTINUE Returned from enumeration callback function to continue enumerating.

Flags for enumerating devices

OAEDFL_ALLDEVICES All installed devices, whether currently attached or not.
OAEDFL_ATTACHEDONLY Only devices that are attached to the system.
OAEDFL_COINACCEPTOR Only Coin Acceptor(door) devices.
OAEDFL_JOYSTICK Only Joystick devices.
OAEDFL_TRACKBALL Only Trackball devices.
OAEDFL_GCICARD Only Game Controller Interface cards.

5.2 Error Return Codes

OA_OK
The operation completed successfully.

OA_NOTATTACHED
The device exists but is not currently attached.

OA_BUFFEROVERFLOW
The device buffer overflowed. Some input was lost.

OA_PROPNOEFFECT
The change in device properties had no effect.

OA_NOEFFECT
The operation had no effect.

OA_POLLEDDEVICE
The device is a polled device. As a result, device buffering will not collect any data and event notifications will not be signalled until GetDeviceState is called.

OA_DOWNLOADSKIPPED

The parameters of the effect were successfully updated by IOAADEffect::SetParameters, but the effect was not downloaded because the device is not exclusively acquired or because the OAEP_NODOWNLOAD flag was passed.

OA_EFFECTRESTARTED
The parameters of the effect were successfully updated by IOAADEffect::SetParameters, but in order to change the parameters, the effect needed to be restarted.

OA_TRUNCATED
The parameters of the effect were successfully updated by IOAADEffect::SetParameters, but some of them were beyond the capabilities of the device and were truncated.

OA_TRUNCATEDANDRESTARTED
Equal to OA_EFFECTRESTARTED | OA_TRUNCATED.

OAERR_OLDOAADVERSION
The application requires a newer version of OAAD.

OAERR_BETAOAADVERSION
The application was written for an unsupported prerelease version of OAAD.

OAERR_BADDRIVERVER
The object could not be created due to an incompatible driver version or mismatched or incomplete driver components.

OAERR_DEVICENOTREG
The device or device instance or effect is not registered with OAAD.

OAERR_NOTFOUND
The requested object does not exist.

OAERR_OBJECTNOTFOUND
The requested object does not exist.

OAERR_INVALIDPARAM
An invalid parameter was passed to the returning function, or the object was not in a state that admitted the function to be called.

OAERR_NOINTERFACE
The specified interface is not supported by the object

OAERR_OUTOFMEMORY
The OAAD subsystem couldn't allocate sufficient memory to complete the caller's request.
An undetermined error occured inside the OAAD subsystem OAERR_GENERIC

OAERR_UNSUPPORTED
The function called is not supported at this time

OAERR_NOTINITIALIZED
This object has not been initialized

OAERR_ALREADYINITIALIZED
This object is already initialized

OAERR_NOAGGREGATION
This object does not support aggregation

OAERR_OTHERAPPHASPRIO
Another app has a higher priority level, preventing this call from succeeding.

OAERR_INPUTLOST
Access to the device has been lost. It must be re-acquired.

OAERR_ACQUIRED
The operation cannot be performed while the device is acquired.

OAERR_NOTACQUIRED
The operation cannot be performed unless the device is acquired.

OAERR_READONLY
The specified property cannot be changed.

OAERR_HANDLEEXISTS
The device already has an event notification associated with it.

E_PENDING
Data is not yet available.

OAERR_INSUFFICIENTPRIVS
Unable to IOAADConfig_Acquire because the user does not have sufficient privileges to change the joystick configuration.

OAERR_DEVICEFULL
The device is full.

OAERR_MOREDATA
Not all the requested information fit into the buffer.

OAERR_NOTEXCLUSIVEACQUIRED
The operation cannot be performed unless the device is acquired in OASCL_EXCLUSIVE mode.

OAERR_NOTBUFFERED
Attempted to read buffered device data from a device that is not buffered.